UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

DEPARTAMENTO DE MATEMÁTICAS

TESIS DOCTORAL

ALGORITMOS DE DESREFINAMIENTO EN MALLADOS ESTRUCTURADOS BIDIMENSIONALES

ANGEL PLAZA DE LA HOZ

Las Palmas de Gran Canaria, marzo de 1993

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

DEPARTAMENTO DE MATEMATICAS

PROGRAMA DE DOCTORADO:
METODO DE LOS ELEMENTOS FINITOS EN LA INGENIERIA

TESIS DOCTORAL

ALGORITMOS DE DESREFINAMIENTO EN MALLADOS ESTRUCTURADOS BIDIMENSIONALES

A. Plaza de la Hoz

1993

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

DOCTORADO EN MATEMATICAS

DEPARTAMENTO DE MATEMATICAS

PROGRAMA DE DOCTORADO: EL METODO DE LOS ELEMENTOS FINITOS EN LA INGENIERIA

ALGORITMOS DE DESREFINAMIENTO EN MALLADOS ESTRUCTURADOS BIDIMENSIONALES

Tesis Doctoral presentada por Angel Plaza de la Hoz

Dirigida *ex-aequo* por

el Dr. D. Luis Ferragut Canals

el Dr. D. Rafael Montenegro Armas

El Director | El Director | El Doctorando

Las Palmas de Gran Canaria, a 1 de Marzo de 1993.

Abstract

The traditional approach to the solution of time-dependent problems using adaptive finite element methods has been revealed very useful to solve this kind of problems. However, in the topics of structured meshes, if only a local refinement is used to approach the solution at each time event, if the refined areas change with the time, the appearance of a large number of nodes creates a serious difficulty. Many of these nodes -though necessary in any past time- are useless in the present moment.

As it is well known, the number of unknowns of the associated algebraic system to the finite element method is about the number of nodes of the mesh multiply by the number of degrees of freedom/node of our problem. So, an increment in the number of nodes in the mesh implies an increase in the number of equations of the system to be solved. Thereby in the topics of structured meshes it seems necessary to develop a derefinement algorithm able to remove dupe nodes, to get a good approximation of the numerical solution obtained in the previous time step and to be combined with a local refinement for the following time step. We have used triangular elements with three nodes and a version of the 4-T algorithm of Rivara at the refining. The derefinement algorithm can be understood as the inverse algorithm of the refinement one.

With this combination (refinement and derefinement), that we call readaptive process, we get families of sequences of structured meshes more flexible than those obtained by local refinement only and with the advantage that the number of equations does not increase so much during the whole evolutive process.

Moreover, the fact of using nested grids enables us to use the multigrid method in order to solve the equation system associated to the finite element method.

In the Thesis there are four main sections as follows. After introduction, in chapter 2 a readaptive process is described in a general manner. Some

definitions, relevant properties and the data structure used are followed by the description of the derefinement algorithm.

Some numerical results are given in the third section. We present examples on some time-dependent problems and also in stationary problems. The numerical solution for a Poisson's test-problem with a time-dependent function on the second member is presented. After, the solution of an evolutive convection-diffusion problem with dominant convection is given. In this last case, we prove too the efficiency of the refinement/derefinement algorithm to approach the initial solution of the problem. A previous numerical study is necessary: the semi-implicit formulation used is shown. Afterwards, the error indicators used and the adaptive strategy employed are explained.

In the last section some remarkable conclusions are emphasized and possible lines of research for the future are pointing out.

Resumen

En los últimos tiempos se ha comprobado la utilidad de los métodos adaptativos de elementos finitos para aproximar la solución de problemas dependientes del tiempo. Sin embargo, en el contexto de los mallados estructurados, si el área refinada cambia con cada paso de tiempo, aparece un gran número de nodos, si sólo se utiliza refinamiento local. Muchos de estos nodos, que fueron necesarios en algún paso de tiempo anterior, son inútiles en el momento presente.

Es bien conocido que el número de incógnitas del sistema de ecuaciones asociado al método de los elementos finitos es del orden del número de nodos de la malla, multiplicado por el número de grados de libertad por nodo del problema. De ahí que la aparición de un gran número de nodos –si sólo se utiliza refinamiento local en mallados estructurados– sea una clara dificultad.

En esta Tesis se presenta un nuevo Algoritmo de Desrefinamiento, capaz de eliminar los nodos inútiles, de conservar una buena aproximación de la solución numérica obtenida en el paso de tiempo anterior, e inverso del Algoritmo de refinamiento 4-T de Rivara. Con esta combinación: refinamiento y desrefinamiento, que hemos llamado *proceso readaptativo,* se obtienen familias de secuencias de mallados estructurados bidimensionales más flexibles que aquellos que resultan del uso exclusivo de refinamiento local, con la ventaja adicional de que el número de ecuaciones no crece demasiado durante todo el proceso.

Además el hecho de usar mallas encajadas hace más fácil utilizar el método multimalla para resolver el sistema de ecuaciones asociado.

En la Tesis hay 4 secciones principales. En la introducción, se comentan los métodos adaptativos en general y se resalta la necesidad de desarrollar un algoritmo de desrefinamiento mediante un problema modelo. También se señalan lo objetivos del trabajo.

En el capítulo segundo se explica el algoritmo de desrefinamiento, núcleo de este trabajo y se describe de forma general qué se entiende por un proceso readaptativo. Además se comentan algunas características del algoritmo.

En el tercer capítulo se muestran algunos resultados numéricos. Se presentan ejemplos de problemas estacionarios y también de problemas dependientes del tiempo como un problema de Poisson modelo en el que la función del segundo miembro depende del tiempo. Después se estudia la solución de un problema evolutivo de convección-difusión con término de convección dominante. En este último caso se prueba además la eficiencia del algoritmo de refinamiento/desrefinamiento para aproximar la solución inicial del problema. Como se hace necesario un estudio numérico previo, se muestra la formulación semi-implícita utilizada. También se explican los indicadores de error utilizados y la estrategia adaptativa empleada.

En el capítulo cuarto se enfatizan algunas conclusiones del trabajo de Tesis y, por último, se señalan posibles líneas de investigación para el futuro.

Agradecimientos

En primer lugar agradezco a mis Directores de Tesis, Luis Ferragut y Rafael Montenegro, no sólo el haber trazado las líneas de investigación, sino también el haber impulsado y seguido de cerca, a pesar de las dificultades superadas, todo el trabajo en estos años.

A Luis Alvarez el habernos puesto en contacto con el tratamiento de imágenes, campo en el que esperamos poder aplicar el algoritmo desarrollado.

A José Luis Balcázar y a Juan Carlos Rodríguez del Pino que han ayudado a estudiar la complejidad algorítmica.

También quiero agradecer a la E.T.S.I.T. y al Departamento de Matemáticas de la Universidad de Las Palmas de Gran Canaria el haber facilitado los medios materiales necesarios para elaborar la presente Tesis, y, especialmente, a todos los componentes del grupo de Elementos Finitos, que, de una u otra forma, han ayudado a que este trabajo llegara a feliz término.

Gracias también a la FUNDACION UNIVERSITARIA DE LAS PALMAS (GRAN CANARIA), y, concretamente al Patrocinador: REFINERIA ACEITERA CANARIA, S.A. (RACSA), que ha financiado en parte este trabajo.

A mis padres.

INDICE

1

Introducción

La generación de mallas es uno de los aspectos más importantes en el estudio de las soluciones numéricas de problemas definidos por ecuaciones diferenciales en cualquier tipo de dominios, mediante el método de los elementos finitos. De hecho los malladores adaptativos han merecido un número especial en *International Journal for Numerical Methods in Engineering* (Volumen **32**, Número 4, Septiembre 1991). Allí se dice en el Prefacio: "La capacidad para generar automáticamente y para controlar adaptativamente las discretizaciones usadas en la solución numérica de las ecuaciones en derivadas parciales es muy importante para aplicar con seguridad de éxito las técnicas del Análisis Numérico. La generación automática de mallados y las técnicas de análisis adaptativo han sido áreas activas de investigación y desarrollo durante bastantes años. En los últimos años se ha extendido la idea de que se debe considerar cuidadosamente la interconexión entre ambos aspectos para desarrollar los procedimientos más eficientes de cara a resolver adaptativamente una gran variedad de problemas" [SheWe91].

Este trabajo de Tesis se encuadra en el marco de la generación de una malla óptima bidimensional, la progresiva automatización de un método de elementos finitos y en el marco de los mallados estructurados. El trabajo está estructurado como sigue. En el presente capítulo se habla en general sobre el método de los elementos finitos y se argumenta la necesidad de desarrollar un algoritmo de desrefinamiento en el contexto de los mallados estructurados. Un problema modelo quasi-evolutivo sirve de ejemplo para resaltar esta necesidad. Se termina el capítulo señalando los objetivos propuestos de la Tesis.

El capítulo segundo corresponde al algoritmo de desrefinamiento desarrollado, y puede decirse que es el núcleo del trabajo. En él se justifica la elección particular del algoritmo de refinamiento utilizado, pues el algoritmo de desrefinamiento puede considerarse inverso del de refinamiento. Se exponen las características del código Neptuno [Ferra87a], [Ferra87b]. A

continuación se especifican algunas definiciones previas y propiedades para una mejor comprensión del algoritmo. Después de explicar el algoritmo de desrefinamiento se resaltan algunas de sus características, cómo se completa el algoritmo con otro de compactación de niveles de malla que optimiza el número de niveles involucrados en la resolución de un problema, se señalan las diferencias más importantes con el algoritmo de desrefinamiento de M. C. Rivara [Rivar89], y se define de una forma general lo que hemos llamado procesos readaptativos.

En el capítulo siguiente se exponen algunos resultados numéricos de la aplicación de métodos readaptativos a diversos problemas tanto estacionarios, como quasi-evolutivos y evolutivos. En este último caso se hace un resumen del estudio numérico utilizado. Se puede destacar aquí, que la variedad de problemas a que se ha aplicado con éxito la combinación refinamiento/desrefinamiento, hace esperar que ésta sea una herramienta muy útil para estudiar otro tipo de problemas que puedan plantearse en el futuro.

Finalmente, se resaltan las conclusiones más importantes de la Tesis y se señalan algunas líneas de investigación futuras.

En el apartado de apéndices, se ofrece una nueva idea de refinamiento en mallados encajados sugerida por el algoritmo de desrefinamiento, se recuerdan algunas definiciones y resultados sobre complejidad y eficiencia de algoritmos y, finalmente, se recogen las publicaciones más importantes a que ha dado lugar, hasta el momento, este trabajo.

1.1. Antecedentes

Una clase importante de problemas que aparecen en física e ingeniería se puede encuadrar en el siguiente marco variacional abstracto:

$$"Hallar\ u \in V\ tal\ que:\quad a(u,v) = f(v)\quad \forall\, v \in V"$$

donde V es un espacio de Hilbert, $a:\ V \times V \to \mathrm{R}$ es una forma bilineal continua y elíptica, y $f: V \to \mathrm{R}$ es una forma lineal y continua. El Teorema de Lax-Milgram asegura entonces la existencia y unicidad de la solución.

En la mayor parte de los casos prácticos no se puede encontrar la solución exacta del problema anterior; se buscan entonces soluciones aproximadas.

La aproximación general de Galerkin consiste en construir subespacios V_h de V de dimensión finita y resolver el siguiente problema aproximado:

$$"Hallar\ u_h \in V_h\ tal\ que:\quad a(u_h,v_h) = f(v_h)\quad \forall\, v_h \in V_h"$$

que equivale a la resolución de un sistema algebraico lineal de ecuaciones.

Ejemplos típicos de espacios V que aparecen en las aplicaciones son $H^1(\Omega)$, $H_0^1(\Omega)$, $H^2(\Omega)$, $H_0^2(\Omega)$, etc. Los problemas de contorno asociados a ecuaciones en derivadas parciales elípticas de 2º a 4º orden lineales son ejemplos que se pueden resolver siguiendo el esquema abstracto anterior.

El método de elementos finitos, en su forma más sencilla, es un método específico para construir subespacios de dimensión finita de V. Por otra parte, la aplicabilidad del método no se limita a los problemas mencionados sino que se extiende a problemas parabólicos, hiperbólicos, no lineales, etc.

El punto de partida, y a su vez el aspecto más característico del método de elementos finitos, es la subdivisión del dominio Ω, en el que está planteado el problema a resolver, en subdominios K mediante, por ejemplo,

una *triangulación* τ del mismo, de modo que se cumplan las siguientes propiedades:

1. $\overline{\Omega} = \bigcup_{K \in \tau} K$

2. $\forall K \in \tau$, K es cerrado y su interior $\overset{\circ}{K}$ es no vacío

3. Para cada par $K_1, K_2 \in \tau$, $\quad \overset{\circ}{K}_1 \cap \overset{\circ}{K}_2 = \varnothing$

Finalmente, un elemento finito, siguiendo el formalismo introducido en Ciarlet [Ciarl78], está caracterizado por una terna (K, P, Σ) donde

i) K es un conjunto cerrado de $\boldsymbol{R}^d$ de interior no vacía y frontera ∂K lipschitziana, donde d es la dimensión del dominio Ω.

ii) P es un espacio de funciones reales definidas sobre K

iii) Σ es un conjunto finito de formas lineales independientes definidas sobre P.

En esta tesis se ha utilizado el elemento finito más sencillo definido por:

K son triángulos

P es el espacio de polinomios de grado menor o igual que uno

Σ es el conjunto de las tres formas lineales que asignan a cada función de P su valor en los vértices del triángulo respectivamente.

Este es un ejemplo de elemento finito de Lagrange, en los que las formas lineales de Σ nos dan el valor de la función en puntos característicos de K. En la literatura es usual designar estos puntos con el nombre de nodos. En el ejemplo anterior, los nodos coinciden con los vértices de los triángulos.

El número y colocación de los nodos introducidos en el dominio inicial, o lo que es lo mismo, el tamaño y colocación de los subdominios que podemos llamar elementos finitos, es muy importante. Ambos aspectos

determinan el grado de exactitud de la solución numérica que hallaremos. Si se reduce el tamaño de los elementos, normalmente se obtiene una solución más exacta. Sin embargo reducir el tamaño de los elementos implica necesariamente aumentar el número de nodos, es decir el número de incógnitas del sistema de ecuaciones asociado. Por tanto el menor tamaño de los elementos conlleva mayor coste computacional y de memoria. En cuanto a la colocación de estos nodos en el dominio, que también es importante, se necesita algún tipo de análisis numérico que nos indique en qué zonas es necesario añadir más nodos. Las regiones con altos gradientes: discontinuidades, zonas de alta concentración de cargas o fuerzas puntuales, capas límites o regiones de alto flujo de calor, etc. necesitarán una alta densidad de nodos, mientras que otras, en las que o bien la solución es conocida o bien varía poco, puede ser suficiente una malla muy grosera.

En muchos problemas de ingeniería que simulan situaciones reales es difícil prever cuál será la malla apropiada. Se pueden realizar varios intentos con el fin de obtener la solución más aproximada si se tiene un límite máximo en cuanto al tamaño de los elementos utilizados o sin utilizar un excesivo número de nodos.

Para alcanzar este objetivo: la obtención de soluciones económicas y con cierta exactitud previamente fijada, además de forma automática, es decir sin necesitar varias pruebas antes de encontrar una estrategia óptima -por ejemplo de mallado-, se han hecho y se siguen realizando muchos esfuerzos en la actualidad. Se podrían distinguir como hace Zienkiewicz [ZieZh91] tres aspectos o fases en este proceso:

1) Estimación del error a posteriori mediante procesos económicos y eficientes,

2) Correcta predicción del refinamiento necesario para alcanzar la exactitud prefijada, y, por último,

3) La programación del refinamiento antes indicado.

Estos tres pasos se pueden resumir en dos si suprimimos el hecho de que deben programarse en un cierto código ya existente o no. Es decir, en un proceso adaptativo hay dos fases: análisis del error cometido con la malla anterior y refinamiento de la malla si procede. Los análisis del error a

posteriori nos permiten calcular para cada elemento una cota del error total cometido -en el caso de estimadores de error-, o bien -en el caso de indicadores de error- nos proporcionan un valor por elemento relativo a otros elementos de la malla. Muchos de los primeros trabajos de análisis del error a posteriori fueron debidos a Babuska y Rheinboldt [BabRh78]. Otros trabajos importantes en este aspecto son los debidos a Kelly y otros [KeGaZ83] y a Gago y otros [GaKeZ83]. En la última referencia se ofrecían algunas estrategias sobre el uso de los estimadores de error a la hora de refinar una malla. Zienkiewicz y Zhu señalaban después, en 1987 [ZieZh87], una estimación del error simplificada que no requería el cálculo de saltos de flujo en las caras. Sin embargo el campo del análisis de errores esta aún abierto en muchos problemas en los que no se dispone de estimadores de error, y, además, donde la eficacia de los estimadores propuestos es discutida.

Una vez que se ha obtenido una estimación o indicación del error en cada elemento, se pueden seguir varias estrategias a la hora de adaptar la malla. Las dos posibilidades básicas son el p-refinamiento (o refinamiento en *p*) y el h-refinamiento (o refinamiento en *h*). El p-refinamiento consiste en un aumento en el grado de la aproximación polinomial dentro de cada elemento finito. El refinamiento en *h* significa simplemente una reducción en el tamaño de subdivisión de la malla. El refinamiento en *p*, especialmente cuando se combina con una formulación jerárquica y con una malla suficientemente adaptada, tiene algunas ventajas: es más eficiente, converge más rápidamente. Sin embargo, incorporar un refinamiento en *p* en un código ya existente generalmente implica una re-estructuración del mismo, cuando no una total re-escritura.

Por otra parte, el refinamiento en *h* es más sencillo y se entiende de forma intuitiva. Es fácil de programar en un código y ha sido ampliamente aceptado.

Dentro del refinamiento en cuanto al tamaño de los elementos, caben dos posibilidades: regeneración de la malla o subdivisión de los elementos ya existentes. Podemos llamar a la primera posibilidad remalladores o mallados no estructurados y a la segunda mallados estructurados.

La alternativa de los remalladores implica regenerar completamente la malla, o bien sólo aquellas regiones con alto error. La ventaja de regenerar

toda la malla es que las áreas en las que el error está por debajo del permitido se pueden hacer más groseras: en ellas se pierden nodos, es decir se da una especie de desrefinamiento (véase, por ejemplo, [ZieZh91]). De esta forma, es posible que el número de nodos en la malla refinada sea menor que en malla anterior. En este sentido, el refinamiento en *h* es más flexible en mallados no estructurados que en mallados estructurados o encajados. Entre los remalladores que utilizan elementos triangulares está el avance frontal (por ejemplo [PeVaM87] y [Cross90]), aunque han aparecido trabajos que dejan abierta la posibilidad de utilizar el avance frontal incluso con elementos rectangulares ([ZhuZi91]) si bien con un menor grado de flexibilidad a la hora de colocar los nodos donde sea preciso, y también de convertir mallados no estructurados triangulares en rectangulares automáticamente ([JoSuK91]). Otro mallador de este tipo es el conocido método de triangulación de Delaunay.

Con la subdivisión de elementos, cada elemento que excede el error admisible prefijado es subdivido en elementos más pequeños de alguna forma. En este sentido se pueden citar, por ejemplo, los trabajos de Bank y Sherman ([BanSh80] y [BanSh83]) y de Rivara ([Rivar84a,b,c], [Rivar87], [Rivar89]). Mediante el refinamiento en mallados estructurados se crea un nuevo nivel de subdivisión de la malla cada vez que se refina, pero no se eliminan nodos. Es decir, el número de nodos va en aumento con el número de refinamientos realizados. Esto es particularmente importante en problemas en los que se requieren zonas de refinamiento móviles, como pueden ser problemas dependientes de tiempo: quasi-evolutivos o evolutivos. Muchos de los nodos que se introdujeron en la malla en instantes de tiempo pasados, pueden resultar en el momento actual innecesarios e incluso perjudiciales al aumentar de forma superflua el número de ecuaciones que hay que resolver. Por otro lado, entre las ventajas de los mallados estructurados la principal es que es mucho más fácil utilizar métodos multimalla para resolver el sistema de ecuaciones asociado al método de elementos finitos, con las ventajas de coste computacional que lleva consigo el método multimalla, sobre todo en problemas con gran número de grados de libertad (véase [HacTr82] y [HacTr86]). Además el proceso de refinamiento es más rápido, y, en ciertos problemas, no se necesitan recalcular las matrices elementales de aquellos elementos que no se refinan; esto último es debido a la estructura de datos asociada al proceso de refinamiento en los diferentes niveles de malla. Otra

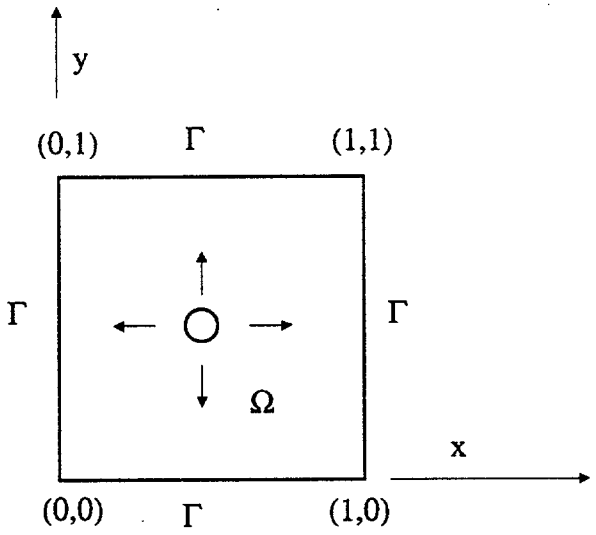
interesante ventaja de los mallados estructurados es que se consigue un cierto control sobre la geometría de las mallas obtenidas, pues ésta dependen de la geometría de la malla inicial. Se evita de esta forma la degeneración del mallado, es decir aparición de ángulos cercanos a cero, que conlleva algunas dificultades numéricas.

En algún caso particular de refinamiento en *h* en mallados estructurados, se ha desarrollado un algoritmo de desrefinamiento, [Rivar89], capaz de eliminar nodos del mallado. Precisamente, el artículo citado de Rivara es el antecedente más próximo del presente trabajo, si bien tiene importantes diferencias con el aquí expuesto. Estas diferencias se comentan en el apartado 2.6.5.

Para hacer más explícita aún la necesidad de desarrollar un algoritmo de desrefinamiento, que dote a los métodos adaptativos en *h* para mallados estructurados de la flexibilidad de que hacen gala los malladores no estructurados, nada mejor que presentar un problema modelo. Consideremos el problema quasi-evolutivo propuesto en el siguiente apartado.

1.2. Un problema modelo

Sea un problema de Poisson, en el que la función del segundo miembro depende del tiempo: una corona circular que se va expandiendo con el tiempo. El dominio es un cuadrado de lado unidad y el paso de tiempo fue fijado al comienzo del mismo, $\Delta t = 1$ (véase fig. 1). Se siguió la estrategia de utilizar 5 refinamientos locales para aproximar la solución en cada paso de tiempo con indicador de Babuska ([BabRh78]) y parámetro de refinamiento $\gamma = 0.6$. En la fig. 2 se pueden observar las primeras mallas y apreciar la evolución del número de nodos.



$$\Delta u = f \quad en \quad \Omega \qquad u = 1 \quad en \quad \Gamma \qquad \text{siendo } f(x,y) = \begin{cases} 0 & si \quad d > r_1(t) \\ 100 & si \quad r_2(t) < d < r_1(t) \\ -10 & si \quad d < r_2(t) \end{cases}$$

$$\text{donde: } d = \left[(x-0.5)^2 + (y-0.5)^2\right]^{1/2},$$

$$r_1(t) = 0.1(1+t) \quad y \quad r_2(t) = 0.1t$$

Figura 1.- Problema modelo. Dominio y condiciones de contorno.

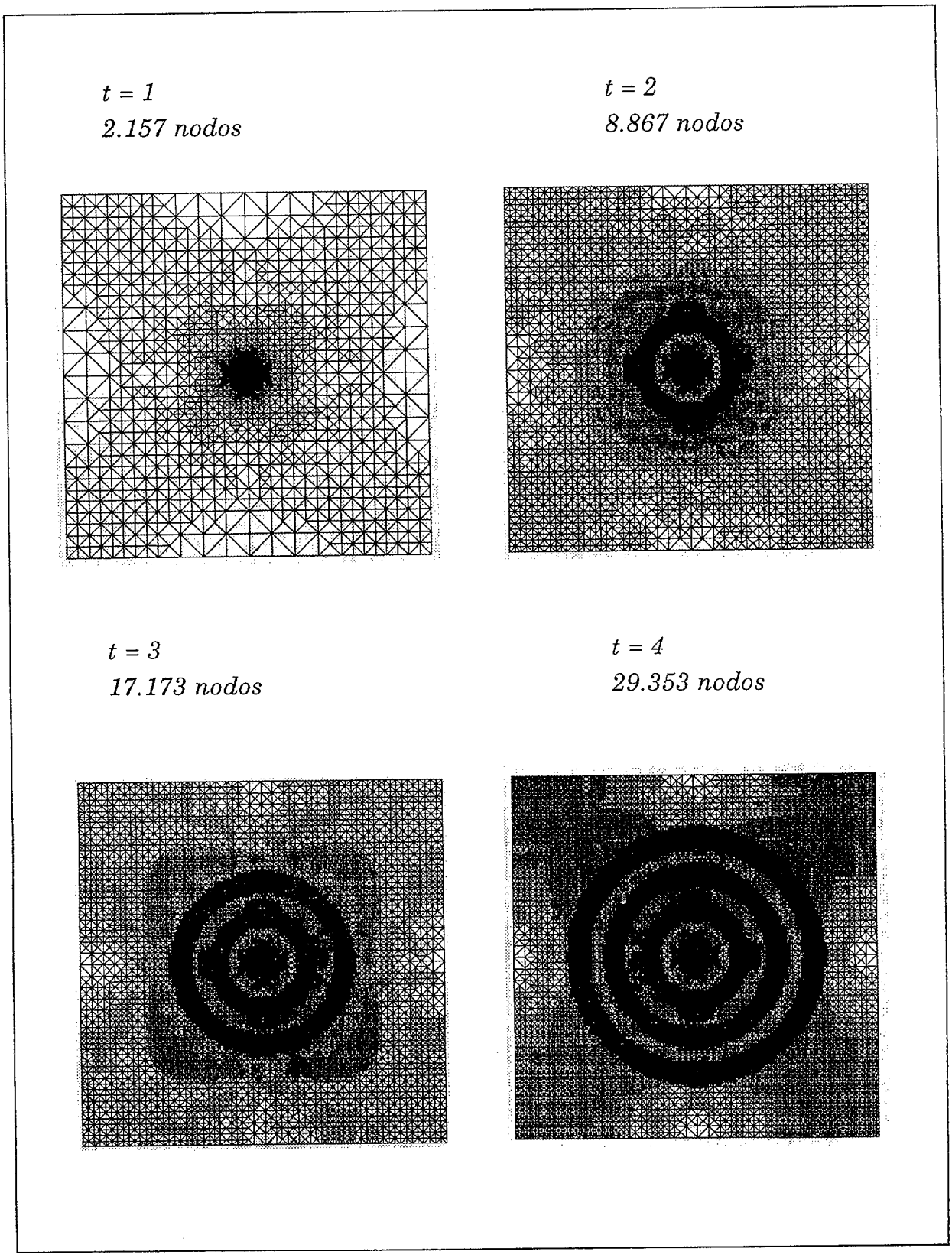


Figura 2.- Cuatro mallas de un problema quasi-evolutivo.

Fácilmente se observa cómo el número de nodos crece de tal manera que tras pocos pasos de tiempo se tienen muchos miles de nodos en el dominio. Resulta, por tanto, impracticable resolver un problema como el propuesto utilizando únicamente el refinamiento local. Podría argumentarse que, puesto que el problema es quasi-evolutivo (sólo hay una dependencia respecto del tiempo en la función del segundo miembro), se podría partir de la malla inicial en cada etapa de tiempo para aproximar la solución. Ciertamente. Sin embargo, este problema puede servir de ejemplo para ilustrar la necesidad de eliminar nodos del dominio, sobre todo en aquellos problemas en los que la solución numérica depende de la solución aproximada obtenida en el instante anterior. Es decir, en problemas en los que la solución anterior es una buena aproximación de la solución en el instante siguiente, como los que se presentan en el apartado 3.3 del presente trabajo.

1.3. Objetivos

Así pues, parece necesario el estudio de un algoritmo desrefinamiento de mallado, al menos en problemas dependientes del tiempo. Nos proponemos los siguientes objetivos en este trabajo de Tesis:

i) Desarrollar un algoritmo de desrefinamiento capaz de:

a) ser combinado con otro de refinamiento en lo que llamaremos proceso readaptativo,

b) eliminar los nodos superfluos del mallado, guardando al mismo tiempo, sobre la malla desrefinada, una buena aproximación de la solución numérica obtenida sobre la malla previa.

ii) Desarrollar algún indicador de desrefinamiento de manera que su computación sea sencilla y eficiente.

iii) Automatizar el proceso de forma que fijando el usuario unos pocos parámetros el código sea capaz de adaptar la malla en cada momento.

iv) Mostrar la eficacia del algoritmo mediante su aplicación a diversos problemas: estacionarios, quasi-evolutivos y evolutivos.

2

El algoritmo de desrefinamiento

En este capítulo se presenta el Algoritmo de Desrefinamiento que se ha desarrollado. En primer lugar se señalan algunos tipos de refinamiento en mallados estructurados triangulares y se resaltan algunas propiedades de ellos. Esto nos permite justificar la elección del algoritmo de refinamiento realizada. En segundo lugar se hacen notar algunas características del código Neptuno tales como la definición de datos físicos y geométricos, el control de la información y de la memoria y cómo se define en el código la estructura multimalla.

Después se introducen algunas definiciones y propiedades de las secuencias de mallas encajadas. Estas propiedades determinan la estructura del algoritmo de desrefinamiento, que se explica en el siguiente apartado. Por último, en el apartado de Observaciones se discuten algunas de sus propiedades. También se completa el algoritmo con la resolución del problema algebraico que conlleva todo desrefinamiento; y se describe un algoritmo de compactación de niveles de malla –el cual puede considerarse como un algoritmo de desrefinamiento de niveles–. Finalmente se comenta la complejidad y eficiencia del algoritmo y se define, de forma general, qué se entiende por un proceso readaptativo.

En lo que sigue trabajaremos con triangulaciones del dominio inicial Ω, que supondremos de frontera $\Gamma = \partial\Omega$ poligonal, o, al menos, regular a trozos. Decimos que τ es una triangulación de Ω, si se cumple:

1) Todo triángulo de τ es un sunconjunto no vacío de Ω.

2) τ está formada por un número finito de triángulos, sea éste $m(\tau)$, entonces se tiene: $\Omega = \bigcup_{i=1}^{m(\tau)} t_i$, donde $\tau = \left\{t_1, t_2, \ldots, t_{m(\tau)}\right\}$.

3) La intersección de dos triángulos no disjuntos de τ, t_i y t_j, es o un vértice o un lado común.

La tercera propiedad exigida se suele expresar diciendo que trabajamos con mallas conformes.

Dada una triangulación τ, sea $\alpha(\tau) = \min_{t \in \tau} \alpha_t$, donde α_t se define como el mínimo ángulo interior del triángulo *t*. Para cada triángulo *t*. consideramos el cociente $\frac{l_t}{h_t}$, donde numerador y denominador son, respectivamente, las longitudes de sus lados más pequeño y más grande. Se define entonces $\delta(\tau) = \min_{t \in \tau} \frac{l_t}{h_t}$. Pues bien, como hace notar Rivara en [Rivar87], los parámetros $\alpha(\tau)$ *y* $\delta(\tau)$ son medidas de la no-degeneración (un triángulo se llama degenerado si el menor de sus ángulos es cercano a cero) y suavidad de la triangulación, esto es, si la transición entre las zonas más groseras y más finas se hace gradualmente o no.

2.1. Algoritmos de Refinamiento

Dentro de los elementos triangulares y de la opción de refinar mediante subdivisión de elementos hay muchas posibilidades. Nuestro principal objetivo es el desarrollo de un algoritmo de desrefinamiento, es decir capaz de eliminar algunos de los nodos introducidos por el refinamiento (local o global). Por tanto, si consideramos el algoritmo de desrefinamiento como el algoritmo inverso del de refinamiento, habremos de elegir cuidadosamente de qué forma serán refinados los elementos. Es decir, la elección del algoritmo de refinamiento es importante .

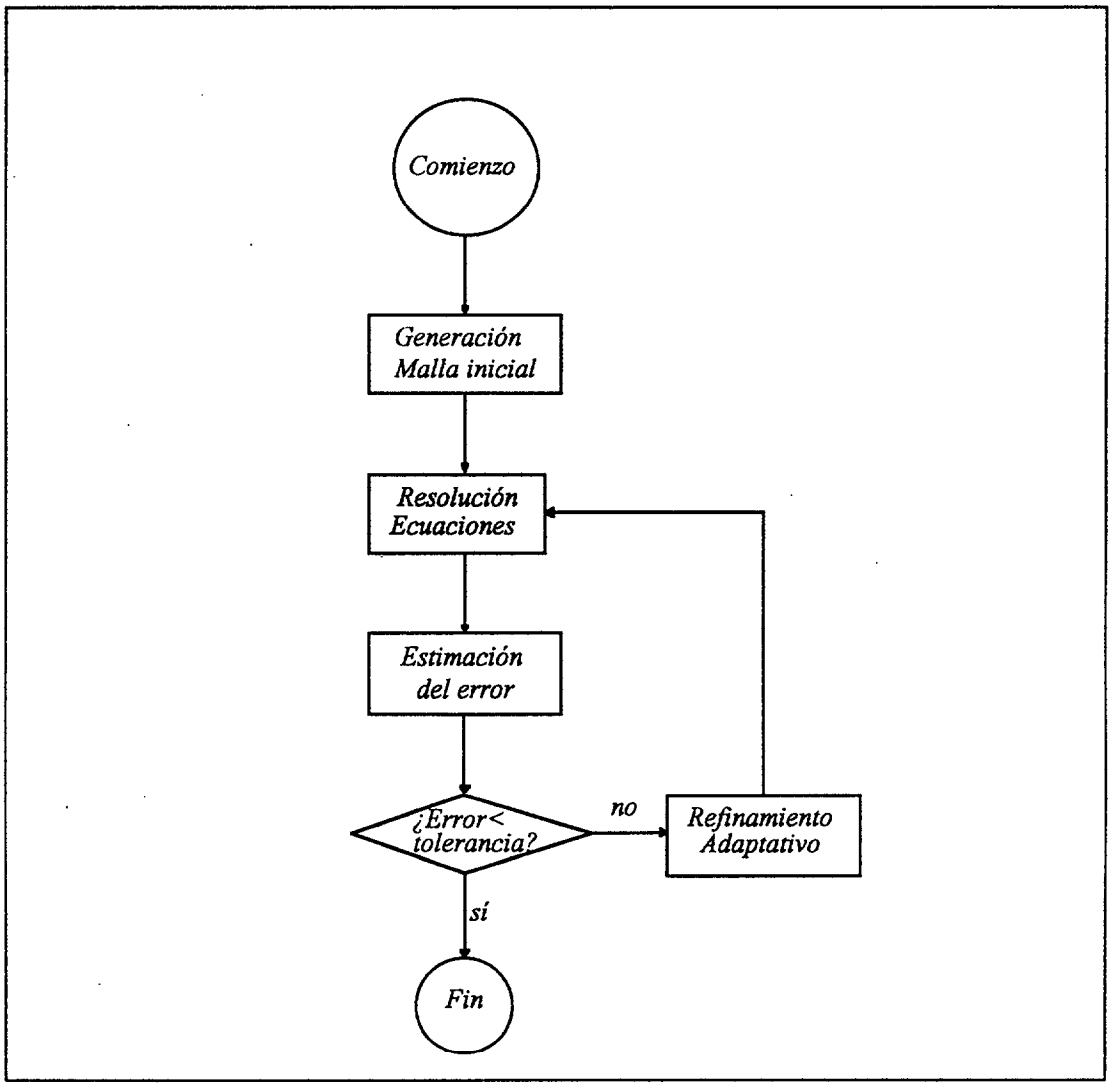


Figura 3.- Diagrama general de un código de Refinamiento Adaptativo.

En general, un código de refinamiento adaptativo sigue el esquema que representa la figura de la página anterior. Hay que señalar, que, en el marco de los mallados estructurados, refinamiento adaptativo significa siempre la inclusión de nuevos nodos en el mallado.

En la figura 4, se muestran algunas posibilidades de división de un triángulo. Entre las formas de división de triángulos más usadas destacan las introducidas por Bank y Sherman ([BanSh80]) y Rivara ([Rivar84a]).

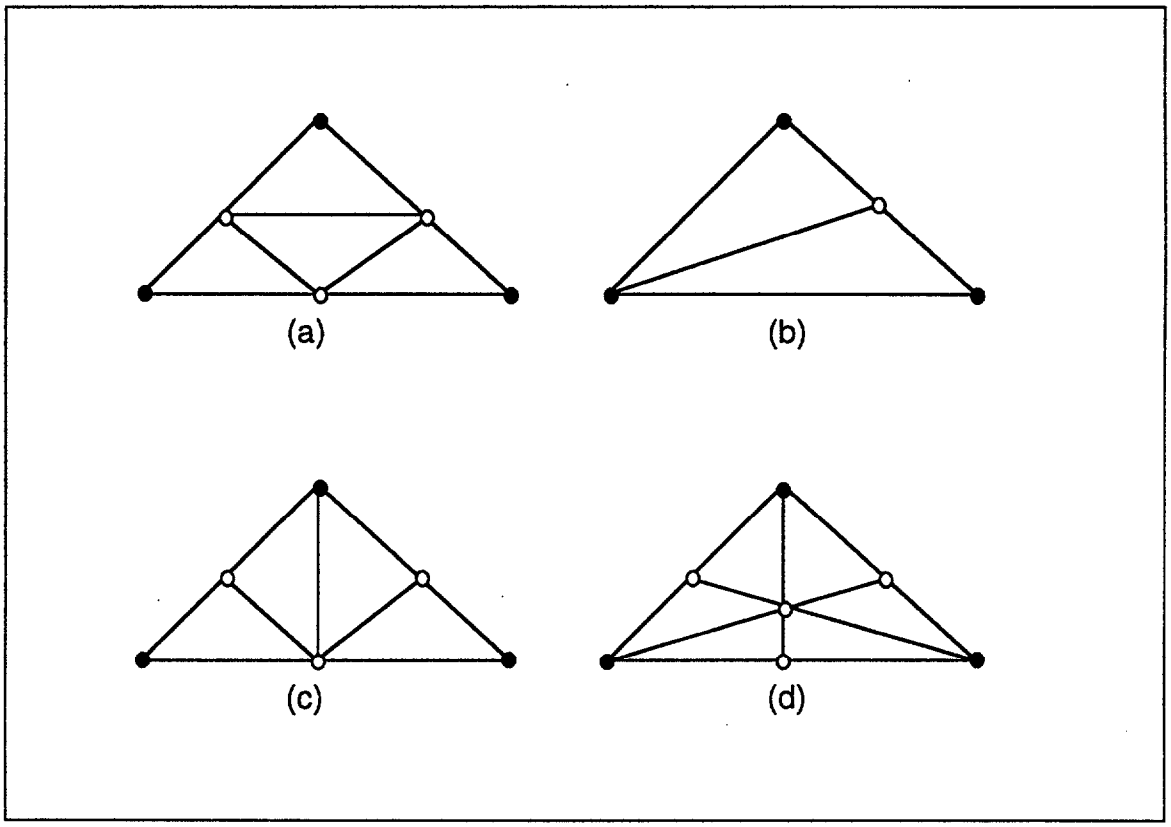


Figura 4.- Diversas formas de dividir un triángulo: (a) División de Bank, (b) Bisección, (c) Algoritmo 4-T de Rivara, (d) otra.

El algoritmo de Bank divide cada triángulo mediante segmentos paralelos a los lados (fig. 4(a)). De esta manera los cuatro triángulos que se obtienen son semejantes al triángulo original. Se mantiene la regularidad en

cuanto a los ángulos. Es decir, si realizamos refinamientos globales mediante este algoritmo y representamos por τ_0 la triangulación inicial y por τ_n la obtenida tras *n* refinamientos globales, se tendrá $\alpha(\tau_0) = \alpha(\tau_n)$.

Sin embargo, si se refina localmente, a la hora de alcanzar la conformidad de la malla, con objeto de asegurar la continuidad de la solución, Bank introduce la llamada "green division" en la que los nodos que se encuentran en el punto medio de alguna cara de algún triángulo de la malla más fina, es decir los llamados *nodos no-conformes*, se unen o al vértice opuesto o a otro vértice no conforme mediante la inclusión de una cara . La fig. 5 representa diversos casos de alcanzar la conformidad según Bank. Esta estrategia tiene el inconveniente de que los triángulos así divididos pierden las buenas propiedades de los divididos regularmente respecto de su forma, aunque, por otra parte, tiene la ventaja de que la conformidad se logra sin añadir ningún nodo adicional.

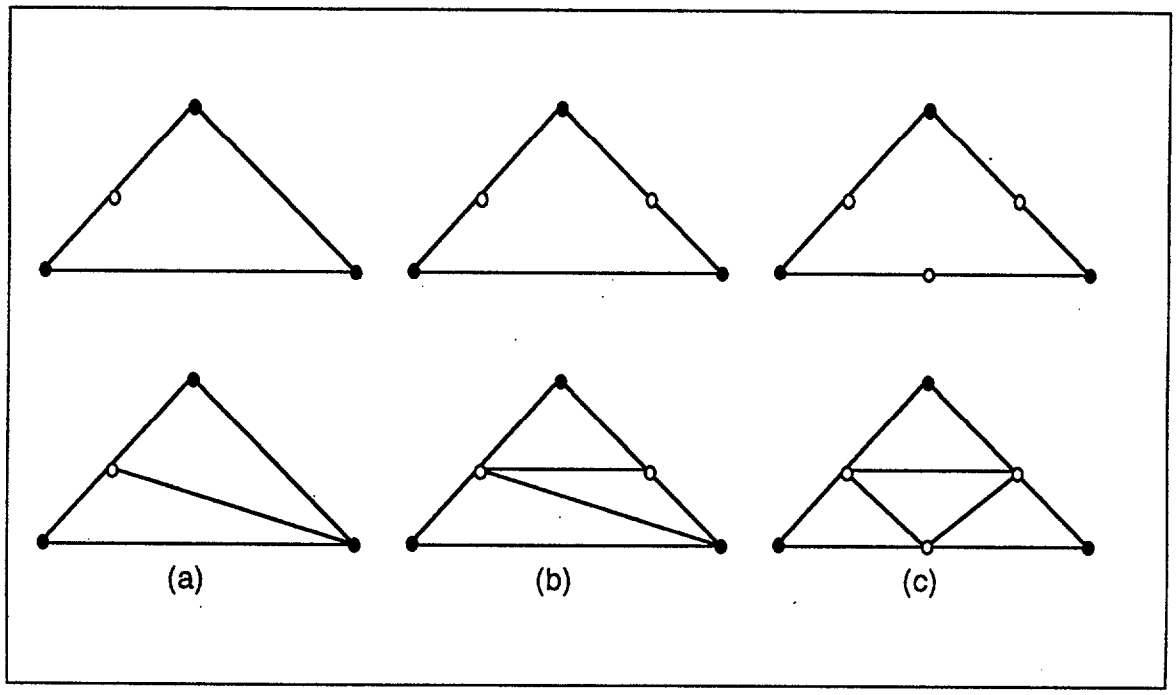


Figura 5.- "Green division" para alcanzar la conformidad: (a), (b) y (c) según tengamos 1, 2 o 3 nodos no-conformes.

Para lograr la conformidad de la malla se podría obrar como hacen Rheinboldt y Metsztenyi en su trabajo sobre mallas no-conformes [RheMe80]. Ellos, para mantener la continuidad de la solución sobre los nodos no-

conformes, imponen la condición de que la solución en estos nodos sea igual a la solución interpolada de otros nodos conformes. Parece, de todas formas, que la aparición de estos nodos no-conformes complicaría el cálculo de las matrices de rigidez elementales al tener que distinguir si un nodo es o no conforme.

Otra posibilidad de división es la bisección (fig. 4(b)), esto es, el triángulo se divide en dos por la mediana a uno de sus lados. En este caso se tiene el problema de que, en general la reiteración de esta forma de refinamiento hace que el ángulo mínimo de la malla tienda rápidamente a cero. Aparecen triángulos degenerados que se comportan mal numéricamente. El mismo problema, agudizado, se tiene si se utiliza la posibilidad de la fig. 4 (d), porque entonces, en cada refinamiento global se asegura que el ángulo mínimo de la malla obtenida es menor o igual que la mitad del ángulo mínimo de la malla anterior. La bisección de un triángulo se puede mejorar, mediante la bisección por el lado mayor. Se tiene entonces el algoritmo 2-T de Rivara o bisección generalizada. La conformidad se lograría del mismo modo que el algoritmo 4-T.

El algoritmo 4-T de Rivara consiste en dividir un triángulo en cuatro del siguiente modo: en primer lugar se divide el triángulo inicial mediante bisección por el lado mayor (bisección generalizada) y después mediante segmentos paralelos a los otros dos lados por el punto medio (fig. 4(c)). Tanto la bisección generalizada, como el algoritmo 4-T tienen muy buenas propiedades en cuanto a regularidad de los elementos a que da lugar. Estas propiedades se basan en trabajos de Rosenberg y Stenger [RosSt75] y Stynes [Styne80] sobre el método de bisección de un triángulo, y están recogidas por Rivara en [Rivar84a]:

1) Sea α_0 el menor ángulo interior de un triángulo *t*. Si dividimos reiteradamente *t* y los triángulos siguientes mediante la bisección por el lado mayor y llamamos α_j al menor ángulo interior de la *j*-ésima triangulación así obtenida –obsérvese que no tiene por qué ser conforme–, entonces se tiene:

$$\alpha_j \geq \frac{\alpha_0}{2}$$

2) Si h_0 es el diámetro –esto es, longitud del lado mayor – del triángulo inicial t, y h_j el diámetro de la malla obtenida tras j refinamientos, entonces existe un entero positivo N, tal que:

$$h_{2j} \leq \left(\sqrt{3}\right)^{min(j,N)} \cdot \left(1/2\right)^{j} \cdot h_0$$

De la propiedad 1) Rivara deduce que, para triangulaciones obtenidas de la aplicación de sus algoritmos 2-T ó 4-T a una triangulación inicial se tiene la misma relación entre sus ángulos mínimos, es decir:

$$\alpha_j \geq \frac{\alpha_0}{2}$$

Además, si notamos por h_i el diámetro de un triángulo $t_i \in \tau_j$, entonces para todo par de triángulos t_1 y t_2 de τ_j, se tiene:

$$\frac{min(h_1, h_2)}{max(h_1, h_2)} \geq \delta > 0$$

Estas dos propiedades le permiten a Rivara hablar de triangulaciones que pertenecen a la misma familia de triangulaciones conformes, ya que las constantes α_0 y δ dependen de la triangulación inicial y caracterizan tales familias de triangulaciones.

Es decir, los ángulos generados por la aplicación reiterada de este algoritmo están acotados inferiormente por una cantidad positiva que no depende del número de iteraciones. De esta forma queda asegurada la no degeneración de los mallados que se obtienen. Y, además la transición entre zonas con pocos nodos y zonas más refinadas es suave. Como señala Thacker [Thack84] son propiedades muy deseables para los generadores de mallas.

Detallando algo más en cuanto a la forma en que se alcanza la conformidad de la malla, hay que resaltar que este algoritmo es preferible a otros, pues los triángulos que aparecen por conformidad son *del mismo tipo* que aquellos que se han obtenido al dividir otros anteriores en cuatro. Es decir, el proceso de hacer conforme un triángulo se puede considerar como una división por 4 interrumpida. Véase más adelante la fig. 6.

Este hecho, tan sencillo, tiene gran importancia desde el punto de vista del desrefinamiento, pues al refinar localmente una malla, cada triángulo sólo podrá dar origen a una de las cuatro posibilidades que aparecen en la fig. 6. Al desrefinar, uno de los triángulos situados a la derecha en la figura, sólo podremos obtener cualquiera situado por encima de él, o bien el triángulo original.

La conformidad en los algoritmos 2-T y 4-T se logra del siguiente modo: si un triángulo tiene algún nodo no-conforme en uno de sus lados que no sea el mayor se introduce otro nodo en el lado mayor y se une con el vértice opuesto; después un segmento paralelo a uno de los lados hace conforme el nodo inicial. Este proceso puede extenderse a otros triángulos, por eso que aparezcan nuevos nodos por conformidad es una desventaja de este refinamiento con respecto al de Bank. Se dice que la zona refinada se extiende por conformidad.

Rivara señala en [Rivar89] que el refinamiento termina siempre en un número finito de pasos. Pero basa su argumento en que, respecto del refinamiento *regular* (en 4) el número de triángulos que pueden refinarse es finito, y, con respecto a la conformidad, señala que ésta sólo se propaga a través de triángulos cuyo lado mayor es más grande, o igual, que el lado mayor del triángulo anterior. Sin embargo, aunque se podría aplicar también a la conformidad el argumento de que sólo se podrán refinar por conformidad un número finito de triángulos, hemos comprobado que a la hora de alcanzar la conformidad pueden aparecer extensiones no deseables, precisamente en mallados de triángulos isósceles o equiláteros, si la comparación entre las longitudes de los lados se realiza sin tener en cuenta los errores de redondeo. En nuestro código esto ha sido subsanado tomando como lado mayor aquel que, salvo una cantidad umbral muy pequeña (del orden de la milésima del lado) es mayor o igual que los otros dos, si esta elección nos permite lograr la conformidad sin necesidad de introducir más nodos.

Por otra parte, la extensión del refinamiento por conformidad hace que la transición entre las zonas más refinadas y las menos refinadas sea suave y, por tanto, que se obtengan refinamientos graduales incluso alrededor de singularidades.

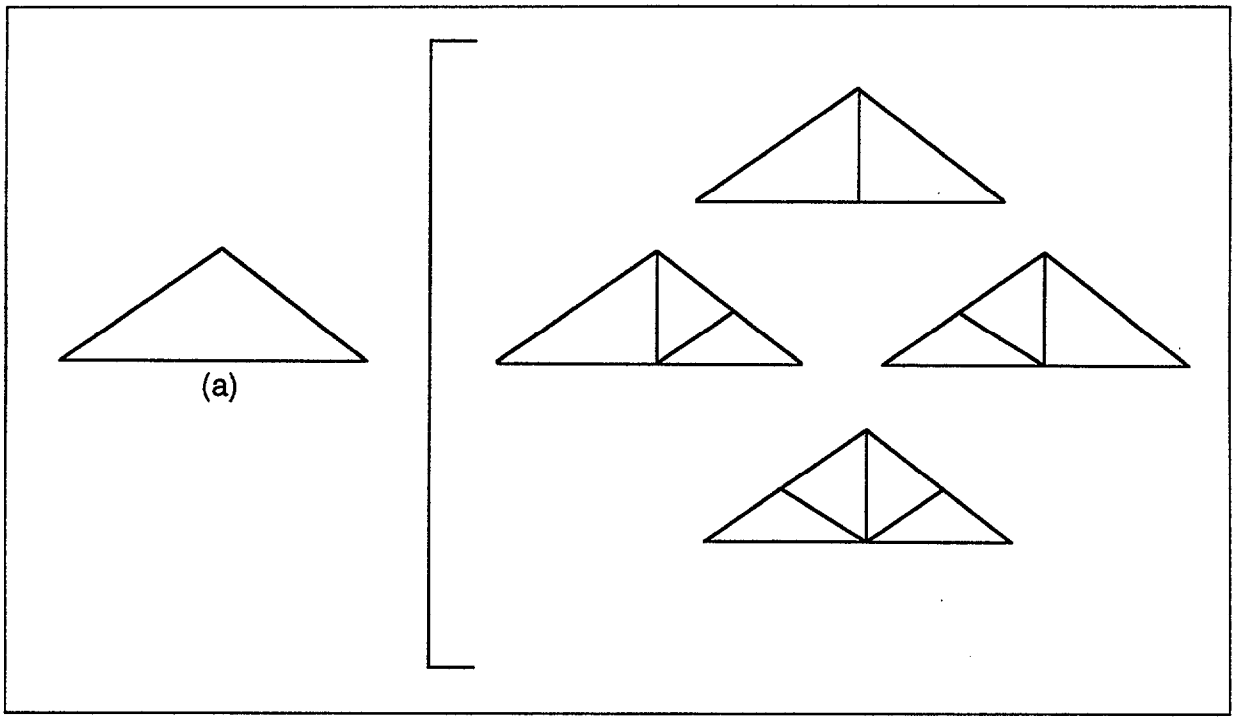


Figura 6.- Las cuatro posibilidades de división de un triángulo.
(a) triángulo original.
A la derecha el triángulo dividido en 2, 3 ó 4 triángulos.

Respecto al algoritmo de Bank y, pensando en el desrefinamiento, se puede comentar que la conformidad se alcanza, en general, de forma distinta a como se realiza la división por cuatro de los triángulos. Se tienen 10 posibilidades distintas de dividir un triángulo si contamos las debidas a la "green division". En la fig. 7 se representan esas distintas posibilidades. Por tanto la programación de un algoritmo de desrefinamiento para el algoritmo de refinamiento de Bank sería más complicada. Además, en el algoritmo de Bank, la transición de la zona más refinada a la menos refinada es más brusca que en el de Rivara, y la acotación del ángulo mínimo tras repetidas aplicaciones del refinamiento no está asegurada.

Por todo lo anterior, hemos elegido el algoritmo 4-T para realizar el refinamiento.

En cuanto a las estrategias de refinamiento, el código Neptuno tiene incorporadas algunas. Por ejemplo se pueden combinar los algoritmos 2-T y 4-T como hace Ferragut en [FeMoW91]: Sea τ una triangulación y N el

número de elementos de τ. Supongamos definido un cierto indicador de error por elemento, es decir, para cada triángulo $t \in \tau$ se conoce el valor de ese indicador de error: η_t. Con objeto de comparar el indicador de error local con alguna medida del error global, consideremos el número η_{opt} definido como:

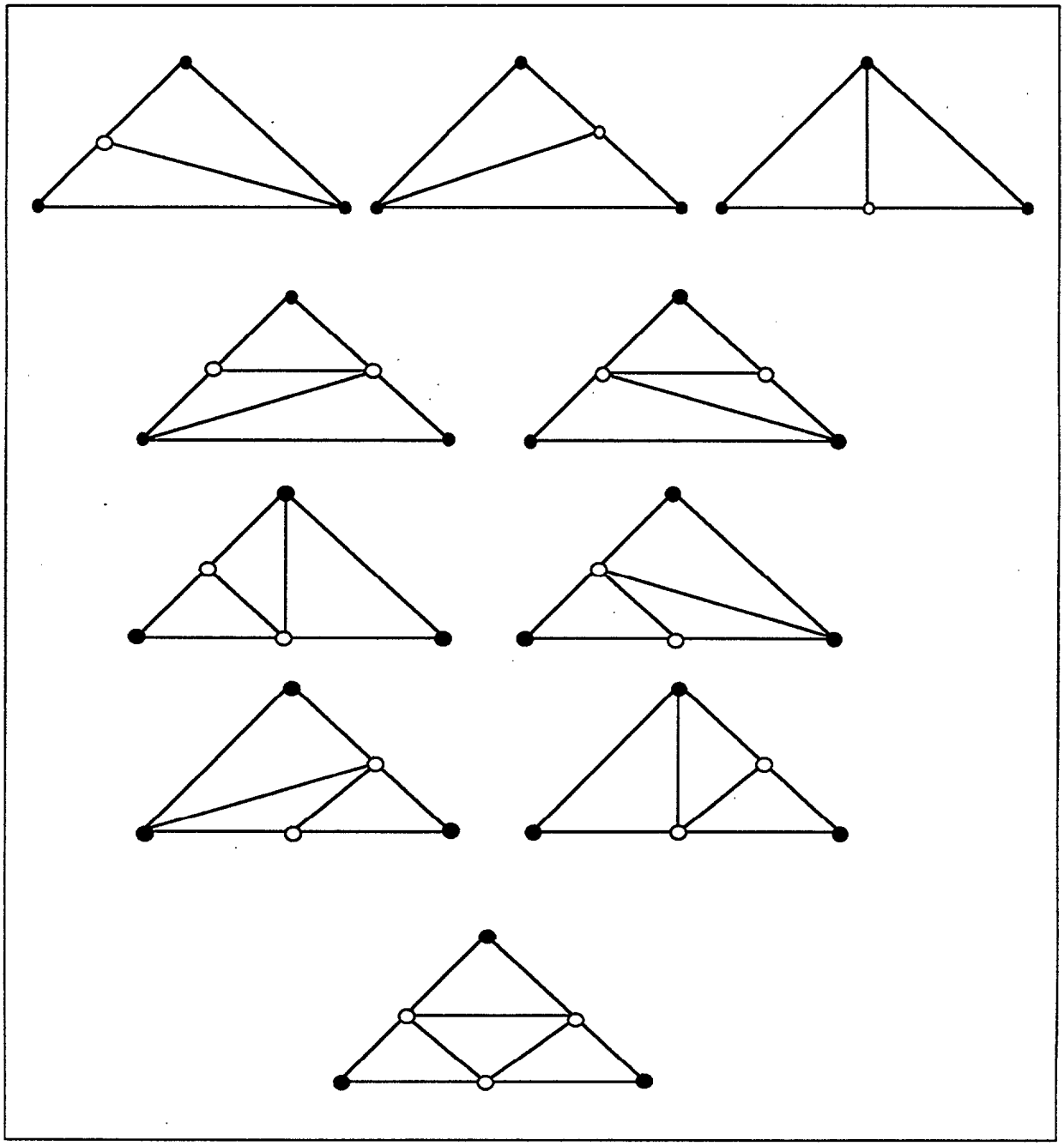


Figura 7.- Las 10 posibilidades de división en un triángulo para alcanzar la conformidad, según Bank.

$\eta_{opt}^2 = \frac{1}{N}\sum_{t\in\tau}\eta_t^2$. Entonces en las primeras etapas de refinamiento se puede seguir la estrategia siguiente:

a) si $\eta_t \geq \gamma \cdot \eta_{opt}$ (con $\gamma \approx 1$) el triángulo se divide en cuatro elementos;

b) en caso contrario, si $\eta_t \geq \beta \cdot \eta_{opt}$ con $\beta = 0.5$, se divide en dos.

En las últimas etapas de refinamiento se refinará sólo de acuerdo con a).

Con este tipo de subdivisión se pretendería alcanzar rápidamente el objetivo deseable de una malla con distribución uniforme de error.

Otra estrategia de refinamiento incorporada al código Neptuno es la siguiente: Sea $\eta_{max} = \max_{t\in\tau}(\eta_t)$. Si $\eta_t \geq \gamma \cdot \eta_{max}$, siendo $\gamma \in [0,1]$ un parámetro elegido por el usuario antes de ejecutar el programa, refinamos en 4 triángulos. Es decir, esta estrategia nos permite refinar los triángulos cuyo indicador de error esté por encima de un cierto tanto por ciento del indicador máximo. Lógicamente, cómo se elija un indicador u otro es de capital importancia para el éxito de una estrategia adaptativa, así como la elección del parámetro γ.

Por último, se puede aplicar también la siguiente estrategia, que no es más que una modificación de la anterior. Con la misma definición de η_{max} que antes, proporcionamos al programa un γ_{min} y un γ_{max}, ambos entre 0 y 1. Además fijamos el número de nodos aproximado que, como máximo, estamos dispuestos a introducir al refinar, sea éste NN_{opt}. Sea, además, NN el número de nodos en un determinado momento en el mallado. Entonces el programa calcula el parámetro de refinamiento γ perteneciente al intervalo $[\gamma_{min}, \gamma_{max}] \subset [0,1]$ más cercano al cociente NN / NN_{opt}. Se consigue de esta forma una mayor libertad en cuanto a los sucesivos refinamientos. Esta variación del parámetro depende del número de nodos NN_{opt} que es estimado *a priori* por el usuario. Esto puede ser útil sobre todo en procesos

en los que el número de nodos puede aumentar o disminuir demasiado, es decir, procesos que hayan incorporado el desrefinamiento.

En cualquier caso, si nos interesa obtener rápidos refinamientos, por ejemplo para hallar una malla relativamente fina, cuando se ha partido de una malla inicial grosera, habrá que elegir valores de γ pequeños. Con $\gamma = 0$ se obtiene un refinamiento global. Para refinamientos más localizados, el parámetro que habrá que tomar estará cercano a la unidad. En problemas que requieren un alto número de refinamientos y cuyo comportamiento no es conocido a priori, la tercera estrategia, con parámetro variable, parece la más adecuada.

2.2. El código Neptuno

El organigrama general del código Neptuno ([Ferra87a], [Ferra87b]) se muestra en la fig. 8. El código ha sido diseñado en forma modular, de este modo se pueden incorporar nuevos problemas fácilmente, así como extenderlo a problemas de mayor número de dimensiones. Precisamente esto ha hecho posible la incorporación del algoritmo de desrefinamiento y algunos otros. Antes de entrar en el desrefinamiento se exponen aquí la estructura de datos y algunos aspectos computacionales que tenía el Neptuno antes de la incorporación del algoritmo de desrefinamiento.

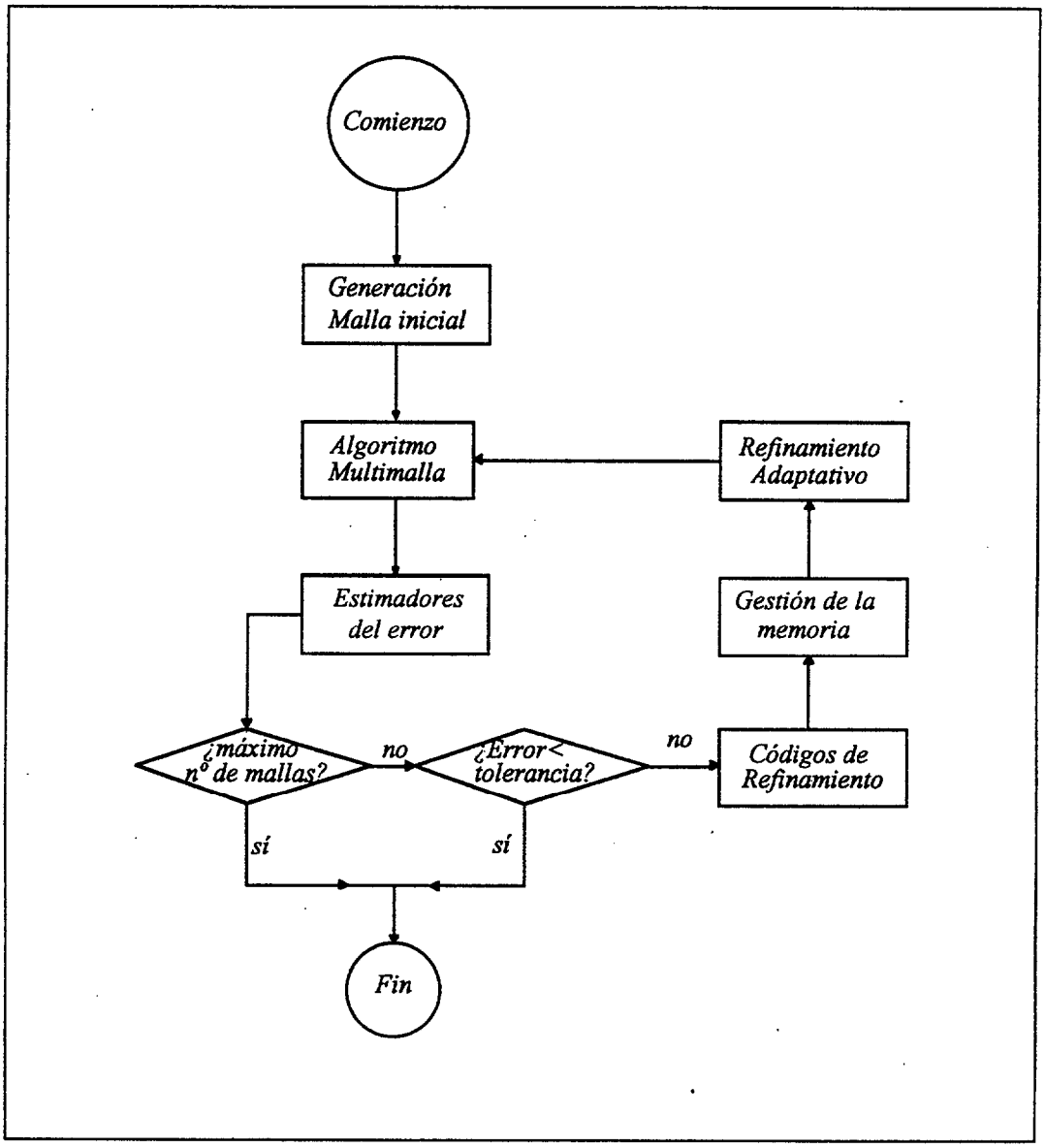


Figura 8.- Diagrama del código Neptuno.

2.2.1. Definición de datos geométricos y físicos

El grupo de datos que caracterizan las propiedades geométricas de la malla están definidas en el código en forma jerárquica, es decir, a partir de los objetos geométricos 0-dimensionales hasta los objetos geométricos bi-dimensionales, según la siguiente secuencia:

a) Definición de nodos por sus coordenadas:

X(1:NDM,1:NUMN), con *NDM* = dimensión del espacio físico
NUMN = número total de nodos

b) Definición de aristas por sus nodos:

IC(1:NEN,1:NUMF), *NEN* = número de nodos de una cara
NUMF= número total de caras

c) Definición de elementos por sus caras

IX(1:NEC,1:NUME), *NEC* = número de caras por elemento
NUME = número total de elementos.

Las fronteras curvas se definen mediante funciones implícitas

$F(x,y) = 0$

La definición de los datos físicos se realiza mediante el número de la función que describe la propiedad física.

2.2.2 Control de la información

Toda la información relativa a los datos geométricos y físicos se controla en el curso de la ejecución mediante dos números clave:

- El número de material, *MA = 1:NUMAT, NUMAT* = número de materiales.
- El número de referencia, *NR = 1:NUREF, NUREF* = número de números de referencia.

A cada elemento se le asigna un número de material que es transferido a sus hijos en el proceso de refinamiento de la malla: *IX(NEC+1,1:NUME).*

A cada nodo y cara se le asigna un número de referencia, que puede ser cero. Cada número de referencia distinto de cero significa que algún tipo de especificación funcional está definida con respecto a ese nodo o cara, por ejemplo una frontera curva, condiciones de contorno de Dirichlet, fuerza externa, etc. El número de referencia es transferido en el proceso de refinamiento a las caras hijas; asimismo un nodo generado sobre una cara adquiere el número de referencia de esa cara. A modo de ejemplo el significado de los números de referencia que aparecen en la fig. 9 es:

NR = 1, significa eje de simetría
NR = 2, condición de Dirichlet
NR = 3 condición de flujo impuesto
NR = 4 condición de Dirichlet y frontera curva

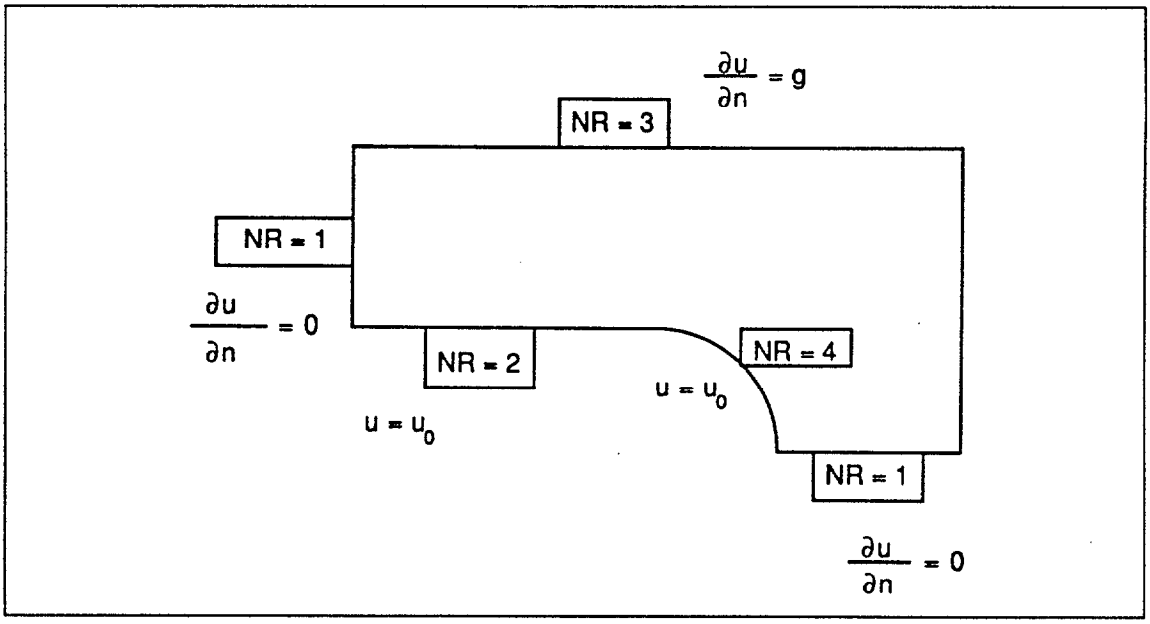


Figura 9.- Números de referencia.

El número de referencia de las caras se almacena en el vector *IC(NEN+1,1:NUMF).* Mientras que el número de referencia de los nodos se almacena en *NUREFN(1:NUMN).*

El significado de cada número de referencia se establece mediante un código de números de referencia que se define mediante dos matrices. La

primera es el vector *IGEOM(1:NUREF)* de dimensión el número total de números de referencia *NUREF*, y que da el número de la función implícita que define la frontera curva; el valor cero quiere decir que este número de referencia no especifica una propiedad geométrica. Las propiedades no geométricas están codificadas, para cada número de referencia *NR*, en la tabla *IREF* siguiente, donde *NDF* es el número total de grados de libertad por nodo:

IREF(1:NDF,NR): > 0 ⇨ nº de la función que define la condición de Dirichlet.

< 0 ⇨ $|IREF(\cdot)|$ es el nº de la función que define una carga puntual.

$= 0$ ⇨ otro tipo de especificación.

IREF(NDF+1,NR): número de función que define el coeficiente de convección.

IREF(NDF+2:2.NDF+1,NR): nº de función de las cargas externas distribuidas sobre la frontera.

El código de los números de material queda definido en la tabla *IMAT(1:20,1:NUMAT)*, donde *NUMAT* = nº total de materiales.

IMAT(1,MA): nº de tipo de elemento.

IMAT(2:20,MA): nº de la función que define una determinada propiedad del material; por ejemplo, componentes del tensor de conductividad, coeficiente de elasticidad, etc.

Finalmente, la estructura de datos se completa con la definición de las funciones, éstas se definen paramétricamente por la tabla:

DFN(1,NCURV) = número de la curva *NCURV*.

DFN(2:20,NCURV) = valor de los parámetros (coeficientes de un polinomio, exponentes, etc.).

2.2.3. Gestión dinámica de memoria

En el proceso de refinamiento, muchas matrices y vectores aumentan de tamaño, de modo que se requiere una gestión dinámica de la memoria. Todos los vectores y matrices se almacenan en un supervector M que se dimensiona en el programa principal. Cuando comienza la ejecución del programa cada bloque tiene espacio suficiente para almacenar, por ejemplo, dos o tres veces el tamaño de la matriz o vector correspondiente a la primera malla. La fig. 10 (a) muestra esquemáticamente el estado del supervector *M* al principio de la ejecución y la fig. 10 (b) después de algunos refinamientos.

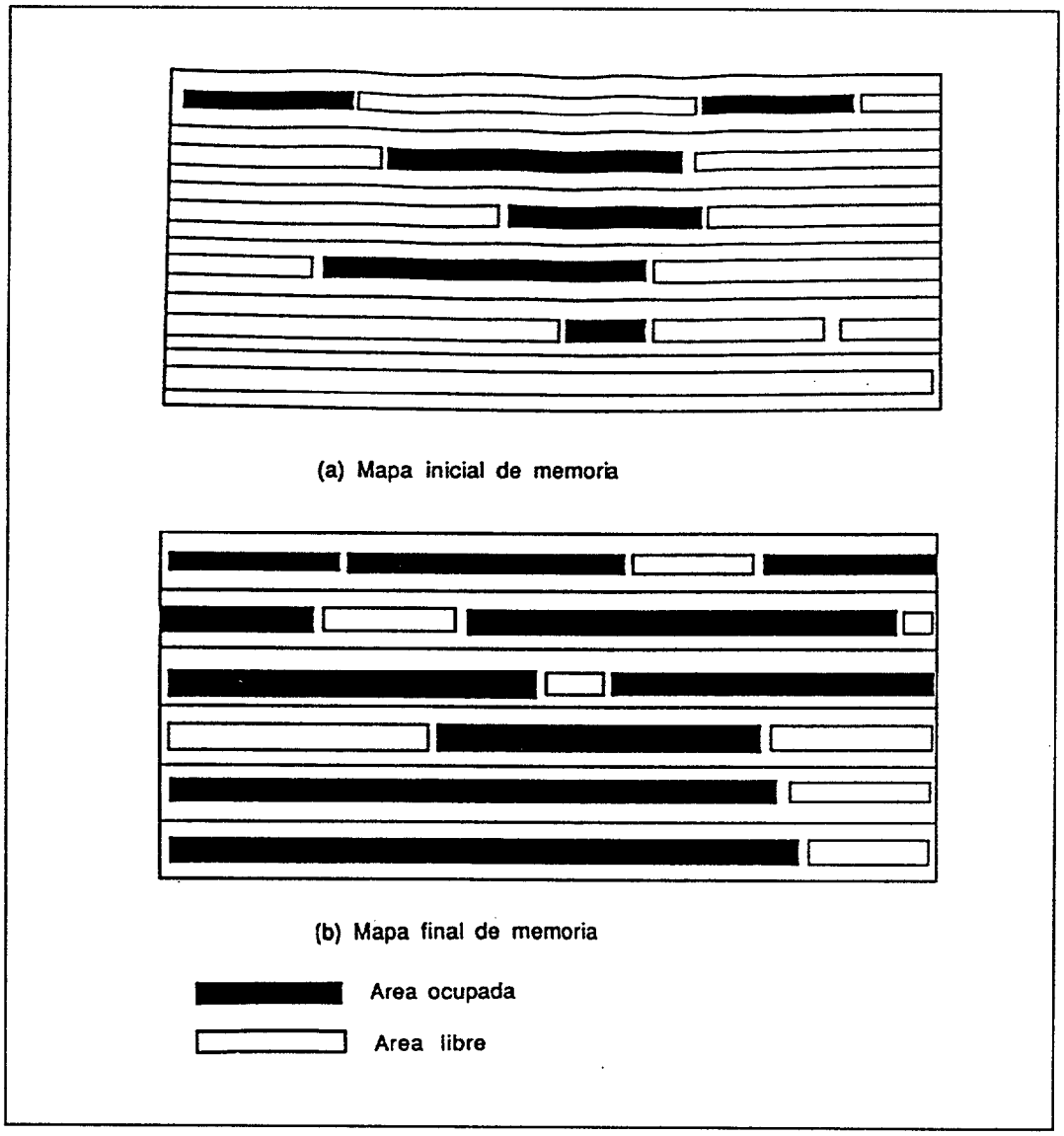


Figura 10.- Manejo de memoria.

Cuando un vector tiene que ser ampliado, el programa comprueba si hay espacio suficiente en el actual bloque; si no lo hay, el programa recoloca el vector detrás de otro vector donde haya espacio disponible, y deja libre el espacio ocupado por él hasta ese instante. El control del direccionamiento se realiza mediante una matriz auxiliar *IM*, donde:

IM(1,IBLOQ) = dirección del bloque *IBLOQ* en *M*.
IM(2,IBLOQ) = número del bloque precedente.
IM(3,IBLOQ) = número del bloque siguiente.
IM(4,IBLOQ) = tamaño del bloque.

2.2.4. Definición de la estructura multimalla

La estructura multimalla queda definida en el código mediante dos vectores, que llamaremos *vectores de estructura*, *IMELEM(1:ISPE)* e *IMFACE(1:ISPF)* donde *ISPE* e *ISPF* se definen así:

$$ISPF = \sum_{j=1}^{n} NF(j) \qquad e \qquad ISPE = \sum_{j=1}^{n} NE(j)$$

siendo *n* el número de niveles en la secuencia de mallas, *NF(j)* = número de caras del nivel j, y *NE(j)* = número de elementos del mismo.

Estos vectores dan para cada elemento (respectivamente para cada cara) de una malla el número global del elemento (o cara). Con la palabra global se quiere decir que esta numeración es independiente de la posición que ocupe ese elemento en cada nivel de malla. Se almacenan los números globales de todos los elementos (y caras) de todas las mallas; si es preciso se puede utilizar una memoria secundaria de acceso directo para almacenar las matrices elementales. También llamaremos a estos vectores *vectores intermedios* porque cada vez que se tenga que hacer un bucle en caras o elementos de un determinado nivel de malla lo haremos a través de estos vectores.

Puede pensarse otro tipo de almacenamiento de datos, sin embargo, con esta estructura de datos, la programación del método multimalla es relativamente sencilla.

En la figura siguiente se representan los vectores de estructura para el caso bidimensional.

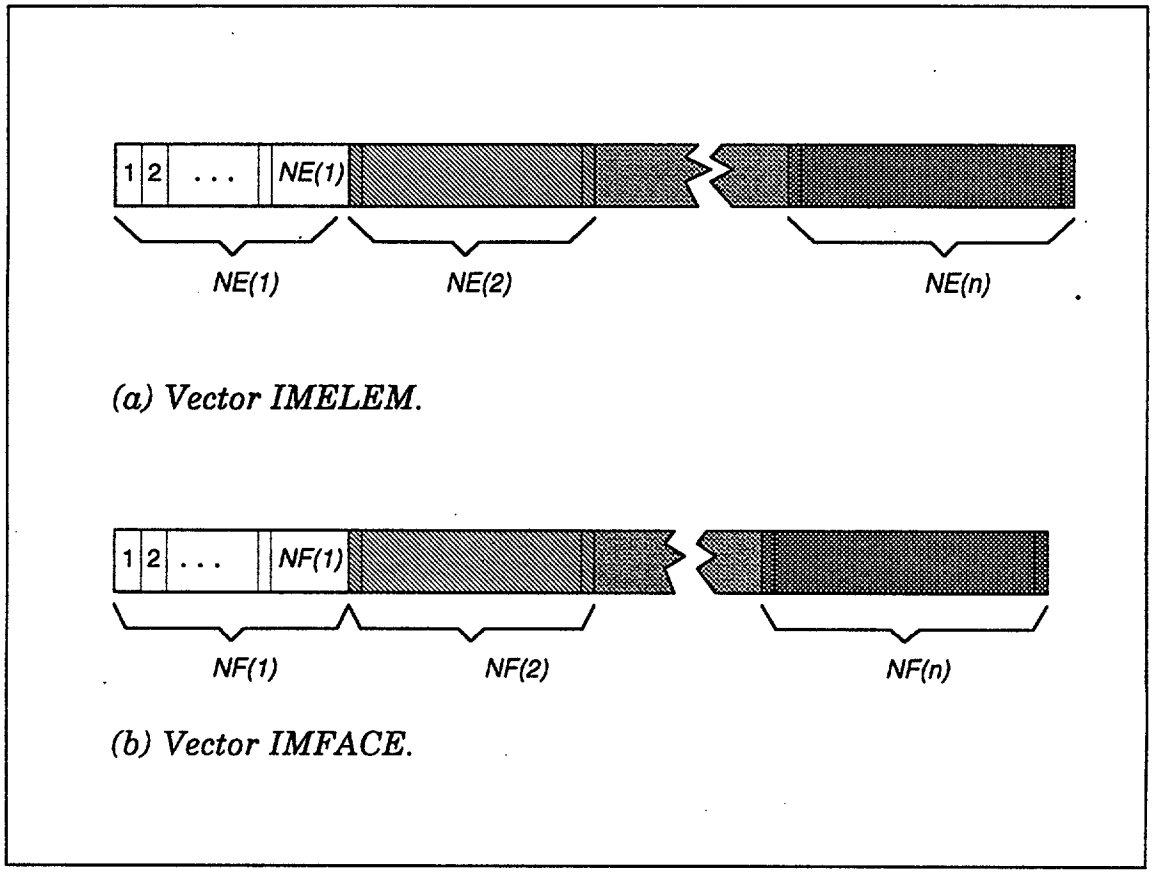


Figura 11.- Los vectores de estructura.

Los dos vectores son análogos. Sólo en la malla inicial la numeración de caras y elementos es consecutiva, desde la unidad hasta el número de caras o de elementos. Además los números *NF(j)* y *NE(j)* crecen con *j*, es decir:

$$NF(j) < NF(j+1)$$
$$NE(j) < NE(j+1)$$

2.3. Definiciones previas y propiedades

Una vez que se han expuesto algunas características del código Neptuno previas a la programación del algoritmo de desrefinamiento, y para fijar la terminología que, por otra parte, ha sido utilizada hasta ahora, damos aquí algunas definiciones relativas a los mallados estructurados. Aunque nos limitaremos a elementos triangulares de 3 nodos, las siguientes definiciones son fácilmente generalizables a otro tipo de elementos y mayor número de dimensiones.

Dadas dos triangulaciones de un dominio plano inicial Ω, τ y τ', diremos que están encajadas y que τ es más fina que τ', o que τ' es menos fina que τ, y escribiremos $\tau \geq \tau'$, o $\tau' \leq \tau$, si se verifican las dos condiciones siguientes:

i) todo nodo de τ' pertenece también a τ.

ii) toda cara (o elemento) de τ' se puede poner como unión (conjuntista) de caras (o elementos) de τ.

A diferencia de la forma en que escribe Rivara esta relación de encajamiento (por ejemplo en [Rivar89]), se puede escribir también con contenidos como $\tau' \subseteq \tau$ (esto es, el contenido tiene el mismo sentido que las desigualdades en la notación anterior). Se observa, de esta forma, que los respectivos conjuntos de nodos de las triangulaciones verifican la misma relación de contenidos. Es decir, si el conjunto de nodos de la malla τ lo representamos por $N(\tau)$, se verifica:

$$\tau' \subseteq \tau \;\Rightarrow\; N(\tau') \subseteq N(\tau)$$

La implicación recíproca obviamente es falsa según la definición de mallas encajadas.

Evidentemente, aunque no lo utilizaremos en lo que sigue, la relación de encajamiento es una relación de orden parcial en el conjunto de todas las posibles triangulaciones sobre un dominio plano inicial.

Pues bien, sea $\mathbf{T} = \{\tau_1 \leq \tau_2 \leq \ldots \leq \tau_n\}$ una secuencia de triangulaciones encajadas, y τ_j una triangulación cualquiera de **T**. Se definen:

1) *Nodo propio y heredado:* Un nodo *N* de τ_j se dice que es *nodo propio de* τ_j si no pertenece a ningún nivel de malla anterior, es decir a ninguna τ_k con $k < j$. En otro caso se dice que el nodo *N* es *heredado en* τ_j. Evidentemente, por la relación de encajamiento entre los conjuntos de nodos, si un nodo *N* es propio de τ_j, entonces es heredado en τ_l, para todo l con $j < l \leq n$.

2) *Cara propia y heredada, y, elemento propio y heredado* se definen de forma análoga a los nodos, es decir una cara (o elemento) se dice *propio* de un determinado nivel de malla, si no pertenece a ningún nivel de malla anterior y *heredado en* ese nivel si fue creado en otro nivel anterior.

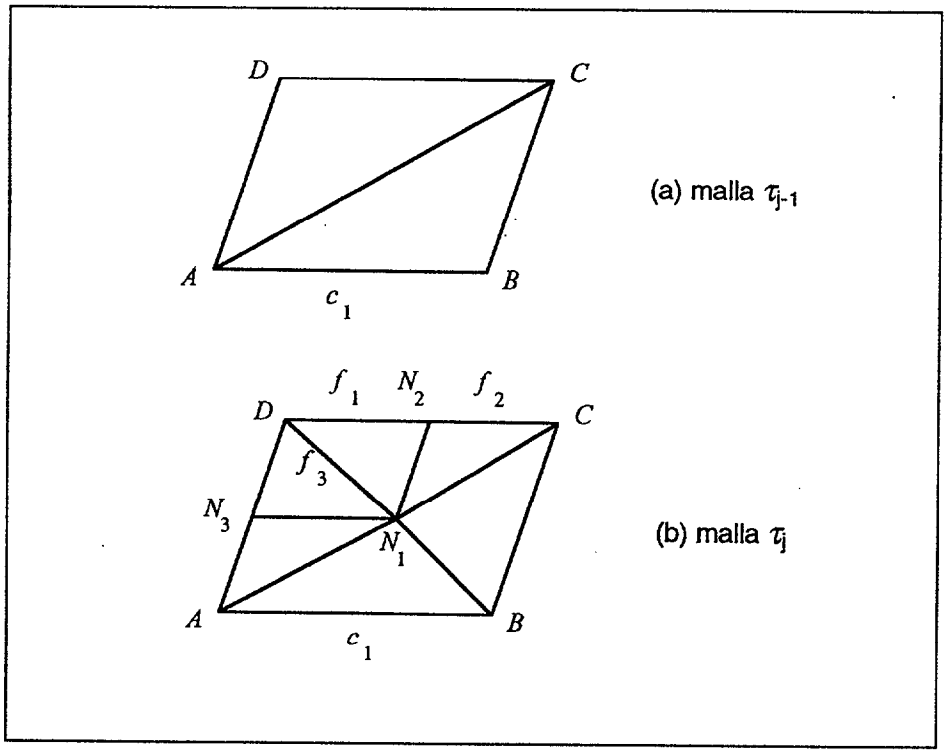


Figura 12.- Ejemplo de refinamiento.

3) *Caras y elementos padres e hijos:* Si al refinar, una cara queda dividida en dos, se dice que la primera es la *cara padre* de estas dos, y éstas

se llaman *caras hijas* de aquella. Según el algoritmo de refinamiento que estamos utilizando, una cara sólo podrá tener a lo más dos caras hijas. Hay que notar que no toda cara tiene cara padre. De forma análoga se definen los elementos padres e hijos. Un elemento tiene, como mucho, cuatro descendientes directos: cuatro elementos hijos. Sin embargo, a diferencia de las caras, todo elemento, excepto los pertenecientes al primer nivel de malla, tienen elemento padre.

4) *Caras internas y externas:* Al refinar un elemento $t \in \tau_j$ aparecen nuevas caras, en el nivel de malla $j+1$. Pues bien, de estas caras, las que no tienen cara padre se llaman *caras internas*, y las que sí tienen, las denominamos *caras externas*. Las primeras se encuentran en el interior del triángulo t las demás están en su frontera.

La fig. 12 muestra un ejemplo de refinamiento. Allí, los nodos N_1, N_2 y N_3 son propios en τ_j y los nodos A, B, C y D son heredados en τ_j. Por otro lado, las caras f_1 y f_2 son externas en τ_j, mientras que f_3 es interna y la c_1 es heredada.

Algunas propiedades de las secuencias de triangulaciones anidadas o encajadas, son las siguientes:

i) todo elemento (o cara) propio de cualquier triangulación es heredado en la triangulación siguientes o tiene sus hijos en ella. Es decir, las familias de caras y elementos relativas a las triangulaciones no están encajadas como ocurre con los nodos.

ii) Por tanto, si un elemento no tiene hijos, pertenece a la malla más fina de la cadena de triangulaciones anidadas.

iii) De ahí que, *sólo aquellos elementos sin sucesores pueden ser eliminados,* a la hora de desrefinar, si se quiere preservar la relación de encajamiento de las triangulaciones.

Esta última propiedad es, al mismo tiempo que muy fácil de comprender, esencial para el algoritmo de desrefinamiento. Es importante destacar que, como se verá más adelante, la malla más fina va cambiando en el proceso de desrefinamiento y por eso tienen sentido las propiedades

anteriores. De esta forma, un elemento perteneciente a un nivel de malla intermedio, será susceptible de ser eliminado al desrefinar, si y sólo si, previamente, han sido eliminados todos sus sucesores.

Veamos ahora cómo hay que modificar la estructura de datos del código Neptuno para la programación del algoritmo de desrefinamiento. Hay que señalar que, mientras que al refinar sólo interviene el nivel de malla más fino y el que se está creando, en un proceso de desrefinamiento es toda la secuencia de mallas anidadas la que puede variarse, así que la estructura de datos será importante, y lógicamente, habrá que almacenar más información que en un proceso simplemente adaptativo. Además, la combinación refinamiento/desrefinamiento implicará algunas modificaciones también del proceso de refinamiento. No vamos a describir con detalle la versión del algoritmo 4-T que utilizamos para refinar. Baste decir que éste opera de forma algo distinta a como lo hace el de Rivara pues establece primero un código de refinamiento que permite buscar espacio en la memoria para almacenar los vectores y matrices que deberán ser ampliadas (ver [Ferra87a]). Sí se señalarán los detalles que se modificarán para combinar el refinamiento y el desrefinamiento.

2.4. Estructura de datos

Hemos de señalar que, en una secuencia de triangulaciones anidadas obtenida por aplicación reiterada de nuestro algoritmo de refinamiento –y también tras cualquier número de refinamientos y desrefinamientos– todo nodo, cara o elemento tendrá el mismo número global *en cualquier nivel de malla* en que se encuentre. Sea, pues, una de tales secuencias de mallas $\mathbf{T} = \{\tau_1 \leq \tau_2 \leq \ ... \ \leq \tau_n\}$ y *NUMN*, *NUMF* y *NUME* los números de nodos, caras y elementos distintos, respectivamente, que se encuentran en la secuencia **T** más los que se encuentran en los *vectores del saco.* (Estos "vectores del saco" serán definidos más adelante. Ahora se puede decir que serán utilizados por el algoritmo de desrefinamiento). Sea, además, *NUMP* el número total de nodos en la secuencia **T**. Se tiene *NUMP* ≤ *NUMN.* Llamemos, para cada nivel τ_j de la secuencia, *NN(j)*, *NF(j)* y *NE(j)* a los respectivos números de nodos, caras y elementos de τ_j; mientras que por *NNP(j)*, *NFP(j)* y *NEP(j)* denotamos los números de nodos, caras y elementos *propios* del mismo nivel de malla.

La estructura de datos que necesitamos añadir a la existente en el código Neptuno se puede resumir como sigue:

a) En cuanto a los *vectores de estructura* o *vectores intermedios,* necesitamos un nuevo vector que almacene la numeración global de los nodos de cada nivel de malla. La razón es sencilla: tras realizar un desrefinamiento, es decir, después de eliminar algunos nodos de una secuencia de mallas, la numeración de los nodos que permanecen no será consecutiva. Así pues, tendríamos dos posibilidades: renumerar los nodos que han quedado, o bien almacenar esa numeración (global) en un nuevo vector. Hemos optado por la segunda posibilidad porque es más sencillo y rápido.

Llamamos al vector de estructura de nodos *IMNODE(1:NUMP).* Este vector está representado en la fig. 13. Se puede señalar aquí, que, puesto que los conjuntos de nodos de mallas encajadas guardan la misma relación de contenidos que las mallas, es suficiente almacenar en *IMNODE* los nodos propios de cada nivel de malla. Por tanto, cada nodo aparece sólo una vez en

este vector, a diferencia de lo que pasaba con los otros dos vectores de estructura *IMFACE* e *IMELEM*.

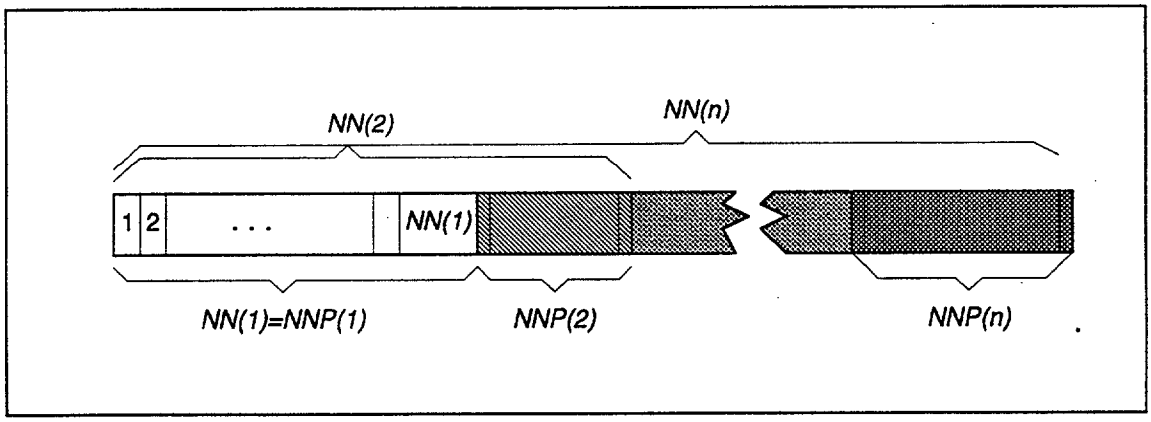


Figura 13.- El vector de estructura IMNODE.

b) Para almacenar la necesaria historia o genealogía de las caras y elementos, introducimos los *vectores de genealogía* que son: *IR[1:3,1:NUMF]* para las caras e *IXH[1:6,1:NUME]* para los elementos. Para cada cara, *IR* nos dice los números de sus caras hijas y el número de su cara padre. De forma similar, *IXH* nos da para cada elemento, el número de los elementos hijos (cuatro como máximo), el número de su elemento padre y, por último, la numeración local del lado mayor. Aquí se puede puntualizar que en el caso de que un elemento tenga 3 hijos, en el cuarto lugar del vector *IXH* se guarda ó "*-1*", ó "*-2*". Se consigue de esta forma distinguir las dos posibilidades de refinamientos en tres elementos: las llamaremos *refinamiento a la izquierda* (y lo señalamos con "*-1*") y *refinamiento a la derecha* (guardamos "*-2*"). Un refinamiento en tres elementos se llama refinamiento a la izquierda si situados en el nodo introducido en el lado mayor del triángulo y de cara al vértice opuesto, el otro nodo introducido, queda a la izquierda. De forma análoga se define el refinamiento a la derecha. Véase la fig. 6 (pág. 23).

c) *Vectores de nivel* o *indicadores de desrefinamiento.* Son tres nuevos vectores: *NODES(1:NUMP)*, *NFACES(1:NUMF)* y *NELES(1:NUME)* para los nodos, caras y elementos respectivamente. Estos vectores almacenan unos indicadores, que en el proceso de desrefinamiento nos dicen si ese nodo, cara o elemento es susceptible de ser eliminado o no. En otra parte del programa, concretamente, en la *compactación de niveles,* guardarán el nivel

de cada uno, es decir, el nivel de malla donde fueron creados, o dicho de otra forma, el nivel en el cual, ese nodo, cara o elemento es propio.

d) En los nuevos *vectores del saco: NNSAC, NFSAC* y *NESAC,* se guardarán la numeración global de los nodos, caras y elementos eliminados al desrefinar. Estos vectores se utilizarán en refinamientos posteriores, y mediante su uso logramos que la numeración global de los nodos, caras y elementos no crezca desmesuradamente tras varios refinamientos y desrefinamientos. La forma en que se calcula el tamaño de estos vectores, que, lógicamente, varía en el proceso, se detallará más adelante.

e) Por último, y con el objeto de facilitar la computación de la condición de desrefinamiento que hemos utilizado, se ha introducido un nuevo vector: *IEX(1:NUMN).* Para cada nodo, *IEX* nos proporciona el número de la cara en la cual ese nodo está en el punto medio; a tal cara la llamaremos la *cara-entorno.* Evidentemente la *cara-entorno* de cada nodo introducido en mallados posteriores al primero es única.

Como puede verse, los requerimientos adicionales de memoria para el algoritmo de desrefinamiento son del orden del número de nodos, ya que el número de caras y de elementos en el mallado es proporcional al número de nodos.

2.5. El Algoritmo de Desrefinamiento

En general, tenemos una secuencia de mallas anidadas:

$\mathbf{T} = \{ \tau_1 \leq \tau_2 \leq ... \leq \tau_n \}$ y queremos obtener otra, tras desrefinar **T**:

$\mathbf{T'} = \{ \tau_1 \leq \tau'_2 \leq ... \leq \tau'_m \}$ es decir que se cumple:

i) $m \leq n$

ii) $\forall\ \tau'_k \in \mathbf{T'},\ \exists\ \tau_i,\ \tau_j \in \mathbf{T}$ tales que $\tau_i \leq \tau'_k \leq \tau_j$

Antes de empezar a hablar del algoritmo de desrefinamiento, se puede decir que las dos condiciones anteriores se pueden tomar como la definición de una relación de encajamiento entre dos secuencias de triangulaciones anidadas definidas sobre el mismo dominio plano inicial. Es decir que si **T** y **T'** son dos secuencias de tales triangulaciones que cumplen las condiciones **i)** y **ii)** anteriores, diremos que **T'** es menos fina que **T** y escribiremos $\mathbf{T'} \leq \mathbf{T}$.

Un esquema básico del algoritmo de desrefinamiento es el que aparece a continuación:

ENTRADA: Secuencia $\mathbf{T} = \{ \tau_1 \leq \tau_2 \leq ... \leq \tau_n \}$

Bucle en niveles de malla, desde la última hasta la segunda. Sea el nivel de malla j: Desde j = n hasta 2, hace:

1. Se recorren los nodos propios de τ_j y se evalúa la condición de desrefinamiento.

2. Se asegura la conformidad de la malla que se está creando minorando la zona que se desrefina.

3.a. Si algún nodo propio de τ_j debe ser eliminado, entonces:

3.a.1. Si algún nodo propio de τ_j debe permanecer, entonces:
Se definen las nuevas conexiones nodales para el nuevo nivel, sea éste τ'_j.

3.a.2. En caso contrario, i.e., si se eliminan todos los nodos propios de τ_j, entonces:
Se elimina el nivel j de los vectores de estructura.

Los cambios se heredan a los siguientes niveles de malla.

Fin del si.

3.b. Si todos los nodos propios de τ_j deben permanecer, entonces:

No se modifica el nivel j: $\tau_j^j = \tau_j$.

Fin del si.

4. Se obtiene una nueva secuencia de mallas que representamos por $T^j = \{ \tau_1 \leq \tau_2 \leq \ldots \leq \tau_{j-1} \leq \tau_j^j \leq \ldots \leq \tau_{n_j}^j \}$.

Fin del bucle en niveles de malla.

SALIDA: Secuencia desrefinada $T' = \{ \tau_1 \leq \tau'_2 \leq \ldots \leq \tau'_m \} =$

$$= \{ \tau_1 \leq \tau^2_2 \leq \ldots \leq \tau^2_{n_2} \}.$$

Dentro del algoritmo de desrefinamiento, podemos distinguir dos problemas: el geométrico y el algebraico. Al primero corresponde el esquema anterior y es el más complicado. El segundo consiste en la reasignación de nuevos números de ecuaciones a los grados de libertad que han quedado en la malla final del desrefinamiento a la vez que se conserva en los nodos que permanecen en el mallado la solución numérica anterior. La fig. 14 muestra un diagrama de flujo del algoritmo de desrefinamiento en su aspecto geométrico.

A continuación, se comentarán más detalladamente cada una de las subrutinas que componen el algoritmo.

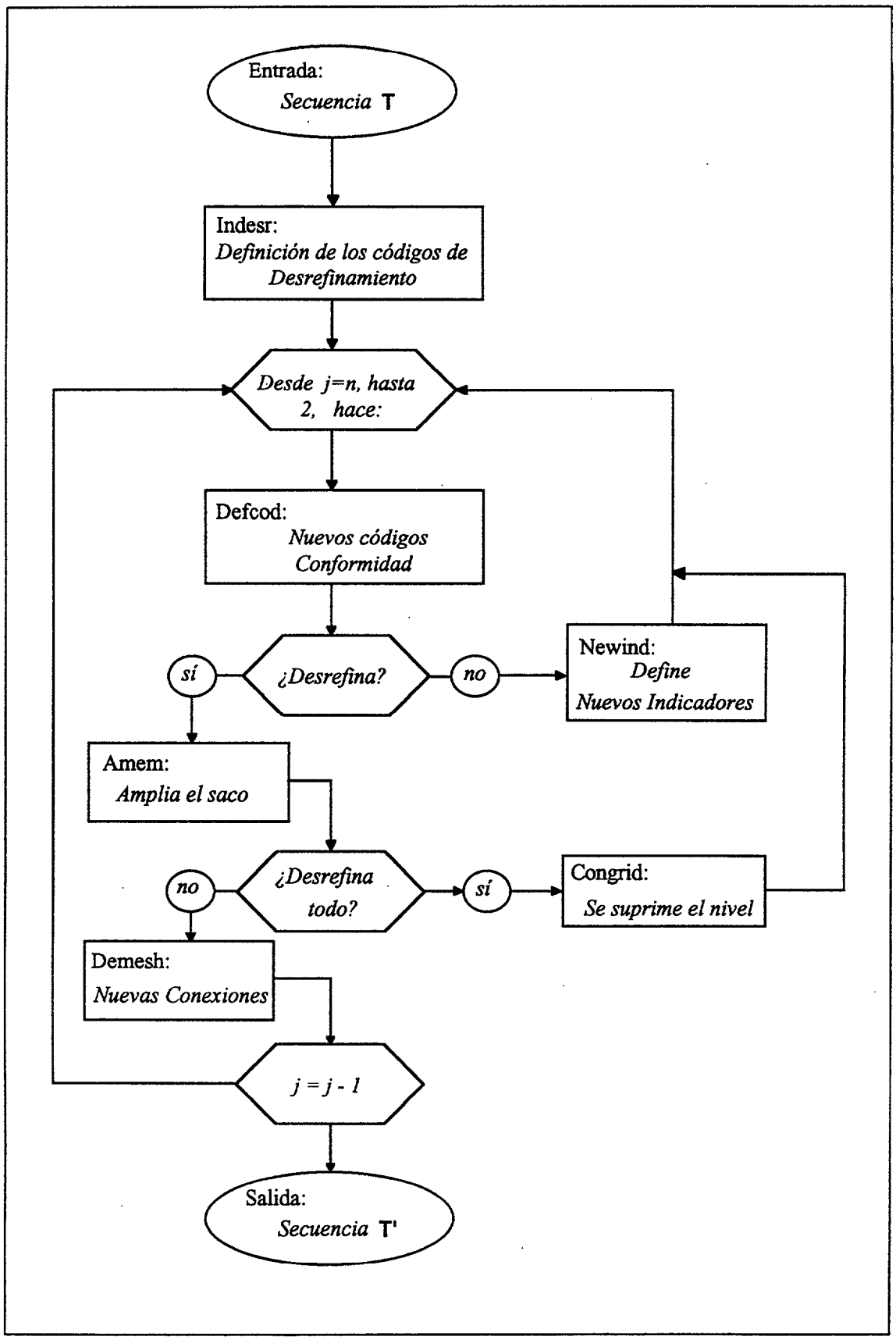


Figura 14.- Diagrama de flujo del Algoritmo de Desrefinamiento.

2.5.1. Inicialización de los códigos de desrefinamiento: Subrutina Indesr

Al comienzo del proceso de desrefinamiento, antes del bucle en niveles de malla, se definen los códigos ó indicadores de desrefinamiento de nodos, caras y elementos. Esto se logra haciendo las siguientes asignaciones en el orden indicado:

Para los nodos: $NODES(N) = 1$, si $N \in \tau_1$
$NODES(N) = 0$, en otro caso.

Para las caras: $NFACES(NF) = 0$, si $NF \in \tau_n$.
$NFACES(NF) = 2$, si $NF \in \tau_j$, para algún j, con $1 < j < n$.
$NFACES(NF) = 1$, si $NF \in \tau_1$.

Para los elementos: $NELES(NE) = 0$, si $NE \in \tau_n$.
$NELES(NE) = 2$, si $NE \in \tau_j$, para algún j, con $1 < j < n$.
$NELES(NE) = 1$, si $NE \in \tau_1$.

Estos indicadores nos dirán si cada nodo, cara ó elemento se puede eliminar (indicador = 0) o no (indicador = 1). Si un indicador no es ni cero ni uno, significará otro tipo de determinación. Por ahora, esto ocurre con las caras y los elementos que son propios en niveles intermedios de malla. Para estas caras o elementos significa que todavía no sabemos nada de ellas respecto del desrefinamiento. Evidentemente los nodos, caras y elementos de la malla inicial han de permanecer en todo el proceso. La diferencia principal entre los nodos y, las caras y elementos es que no hay relación genealógica entre los nodos de diferentes niveles de malla y sí entre las caras y elementos de niveles distintos.

Las tres partes principales del algoritmo son:

1ª) Definición de los códigos de desrefinamiento en cada nivel de malla atendiendo a la condición de desrefinamiento. Aquí también se asegurará que los códigos hagan conforme la malla que se está creando (Subrutina Defcod).

2ª) Gestión dinámica de la memoria, en la que fundamentalmente se amplían los vectores del saco (Subrutina Amem).

3ª) Por último, redefinición de los nuevos indicadores, si no se elimina ningún nodo (Subrutina Newind); supresión del nivel de malla, si es que se eliminan todos sus nodos propios (Subrutina Congrid); o bien, redefinición de las nuevas conexiones nodales (Subrutina Demesh), si el desrefinamiento del nivel de malla es sólo parcial.

2.5.2. Los códigos de desrefinamiento: Subrutina Defcod

Esta subrutina determina si hay o no desrefinamiento en el nivel de malla que le llega, si este desrefinamiento es parcial o total, calcula el tamaño en que deberán ser ampliados los vectores del saco para almacenar en ellos la numeración de los nodos, caras y elementos que serán eliminados y, además, asegura la conformidad de la malla que se creará mediante los códigos de desrefinamiento que se asignan a nodos y caras.

Previamente, se inicializan los indicadores de caras y elementos correspondientes al nivel de malla que se va a desrefinar (τ_j) y al nivel anterior. Esto se hace de forma que los elementos y caras propias del nivel de malla τ_j susceptibles de ser eliminados tengan indicadores igual a cero y sólo ellos.

Inicialización de indicadores de caras y elementos:

Es similar a la que realiza la Subrutina Indesr, pero hay que tener en cuenta que, en general ya hay caras y elementos que han de permanecer: aquellos con indicador de desrefinamiento unidad. Por otra parte basta manejar las caras y los elementos del nivel de malla τ_j y del anterior.

Para las caras: si $NF \in \tau_j$ y $NFACES(NF) \neq 1$, hace:

$$NFACES(NF) = 0,$$

si $NF \in \tau_{j-1}$ y $NFACES(NF) \neq 1$, hace:

NFACES(NF) = 2,

Para los elementos: si $NE \in \tau_j$, y *NELES(NE)* $\neq$ 1, hace:

si NE tiene algún hijo, hace:

NELES(NE) = 1,

en otro caso, hace:

NELES(NE) = 0,

si $NE \in \tau_{j-1}$ y *NELES(NE)* $\neq$ 1, hace:

NELES(NE) = 2.

Hay que hacer notar que es posible encontrar algún elemento de τ_j con indicador distinto de la unidad y que, sin embargo, tenga hijos (en niveles posteriores). Como ese elemento no es susceptible de eliminarse ha de hacerse su indicador igual a 1. Tras esta inicialización de indicadores, las caras y elementos propios de τ_j susceptibles de ser eliminados tendrán indicadores igual a cero.

Se pueden distinguir dos procedimientos principales en esta subrutina: la evaluación de la condición de desrefinamiento y el subproceso en el cual se asegura la conformidad de la malla, en cuanto a los indicadores, (en el caso de que el desrefinamiento sea parcial). Los esquemas de ambos procesos son los siguientes:

Evaluación de la condición de desrefinamiento:

ENTRADA: Nivel de malla τ_j y anteriores: especificaciones geométricas (es decir, conexiones nodales, cara-entorno de los nodos) y algebraicas (esto es, solución numérica en los nodos, números de referencia) e indicadores de desrefinamiento de nodos y caras.

NNP(j) = nº de nodos propios de τ_j.

Para i=1, hasta NNP(j), hace:

N_i = nodo propio i-ésimo de τ_j.

Si N_i es susceptible de ser eliminado:

- Evalúa la condición de desrefinamiento en N_i. (Subrutina Erint).

- Se asignan indicadores de desrefinamiento a N_i, a su cara entorno, a sus caras hijas y a sus nodos extremos.
- Se cuentan los nodos propios que se pueden eliminar en τ_j.

En caso contrario:

- Se asignan indicadores de desrefinamiento a N_i, a su cara entorno, a sus caras hijas y a sus nodos extremos.

Fin del si.

Fin del bucle en nodos propios.

SALIDA: Nuevos códigos en caras y nodos de τ_j y de niveles anteriores. Indicación de si hay o no desrefinamiento, de si éste es parcial o total, y del número de nodos que se pueden eliminar.

Algunos comentarios sobre este procedimiento son:

El que un nodo sea o no susceptible de ser eliminado, nos lo dice el valor de su indicador de desrefinamiento. Si éste es cero, el nodo puede ser eliminado y si es "1", no podrá ser eliminado. Al principio, cuando la malla que le llega a Defcod es τ_n todos los nodos propios tendrán indicador cero, pero en adelante puede que no sea así, incluso puede que ningún nodo sea susceptible de eliminarse. De esta forma se logra evaluar la condición de desrefinamiento en un número de nodos óptimamente mínimo: en cada nivel de malla sólo se toman para evaluar la condición de desrefinamiento los nodos propios, y de estos sólo aquellos susceptibles de ser eliminados.

Esta primera parte de la subrutina Defcod está fuertemente condicionada por la condición de desrefinamiento que se esté utilizando. Las que hemos programado hacen referencia a los nodos. Si la condición fuera relativa a los elementos, el bucle principal habría de ser en elementos.

En las condiciones de desrefinamiento programadas, se compara la solución numérica en cada grado de libertad por nodo con la solución interpolada de la solución en los extremos de su cara-entorno. Hemos utilizado la diferencia relativa y la diferencia absoluta entre estos dos valores. Si esta diferencia es menor que un cierto valor umbral o tolerancia, ε, que fija el usuario en la entrada de datos del problema, el nodo puede ser eliminado y

en caso contrario no. Se puede observar que se comparan dos soluciones aproximadas, no la solución aproximada con la solución exacta, ya que ésta última es en general desconocida. Sin embargo, la idea de la condición de desrefinamiento es que si la solución interpolada es de por sí una buena aproximación de la última solución numérica, nos quedamos con la primera.

En la fig. 15, a la izquierda se muestra un nodo propio K y su cara-entorno con nodos extremos K-1 y K+1. Para decidir si se puede eliminar el nodo K o no, se debe comprobar si para todos los grados de libertad por nodo se cumple la condición:

$$\left|u_h^l(K)-u_i^l(K)\right| \le \varepsilon,$$

o la condición:

$$\left|u_h^l(K)-u_i^l(K)\right| \le \varepsilon \cdot \left|u_h^l(K)\right|,$$

donde $u_h^l(K)$ es la solución numérica en el nodo K correspondiente al grado de libertad l, y $u_i^l(K)$ es la solución interpolada en K, es decir:

$$u_i^l(K)=\frac{u_h^l(K-1)+u_h^l(K+1)}{2}.$$

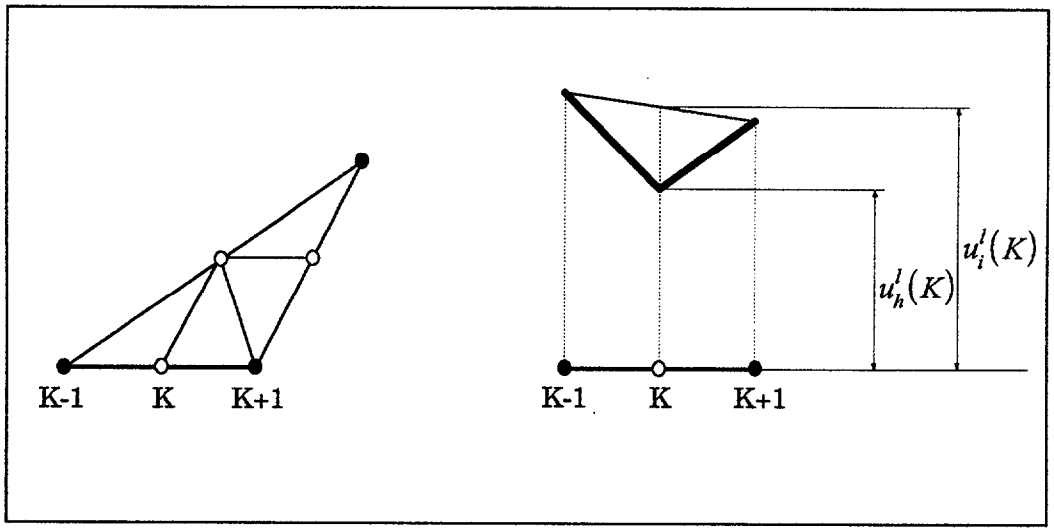


Figura 15.- La condición de desrefinamiento.

Hemos usado la condición de diferencia absoluta cuando tenemos alguna información sobre el rango de los valores que tomará la solución en el dominio, de acuerdo con las características físicas del problema. Sin embargo, si se usa la diferencia relativa, ha de tenerse cuidado con los errores de redondeo en las zonas en las que la solución sea muy cercana a cero.

Si se usa la diferencia absoluta al desrefinar, la solución capturada por la malla desrefinada, $\tilde{u}_h$, y la solución numérica anterior, u_h, difieren en la norma L^∞ menos que ε.

Tras la evaluación de la condición de desrefinamiento, el procedimiento busca alcanzar la conformidad (en cuanto a los indicadores) si el desrefinamiento que se realizará es parcial. Si el desrefinamiento es global o si no desaparece ningún nodo, la conformidad ya queda asegurada pues todos los niveles de malla de la secuencia original eran conformes.

Procedimiento de Conformidad:

ENTRADA: Especificaciones geométricas del nivel τ_j y de los anteriores, indicadores de desrefinamiento de nodos y caras, nº de nodos marcados para eliminar.

NE(j-1)= nº de elementos de τ_{j-1}.
Mientras haya que lograr la conformidad, hace:
    Para i=1, hasta NE(j-1), hace:
    t_i = i-ésimo elemento de τ_{j-1}.
        Si t_i tiene hijos en τ_j,
            Se logra la conformidad local.
        Fin del si.
    Fin del bucle en elementos.
Fin del mientras.
SALIDA: Especificaciones geométricas del nivel τ_j y de los anteriores, indicadores de desrefinamiento de nodos y caras, nº de nodos marcados para eliminar. Indicación de si el desrefinamiento es parcial o total.

A continuación, se explica cómo se alcanza la conformidad local, es decir cómo se logra la conformidad en cada elemento de τ_{j-1} que tenga hijos en τ_j.

Procedimiento de Conformidad Local:

ENTRADA: Elemento t_i, conexiones nodales e indicadores. Número de nodos que se pueden eliminar de τ_j.

N = nodo central del lado mayor de t_i.
Si NODES(N)= 0, entonces:
Si algún otro nodo medio tiene NODES(.)=1, entonces:

- Se hace NODES(N)=1
- Se asignan indicadores =1 a su cara entorno, las caras hijas de ésta y sus nodos extremos.
- Se hace necesario alcanzar la conformidad global.

Fin del si.
Fin del si.

SALIDA: Elemento t_i, conexiones nodales e indicadores. Número de nodos que se pueden eliminar de τ_j. Indicación sobre si hay que alcanzar la conformidad global o no.

Respecto de la forma en que se alcanza la conformidad de la malla desrefinada que se está creando se puede decir lo siguiente:

Con el procedimiento de Conformidad Global se logran asignar indicadores de nodos y caras de manera que más tarde la malla que se creará utilizando esos indicadores será conforme. La manera en que esto se asegura es manteniendo (haciendo su correspondiente indicador *NODES(·)*=1) algunos nodos que de otra forma, y respecto de la condición de desrefinamiento podrían ser quitados. Es decir que el criterio es restrictivo; para alcanzar la conformidad se minora la zona desrefinada. Esta forma de actuar está de acuerdo con el hecho de considerar el algoritmo de desrefinamiento como el algoritmo inverso del de refinamiento. Si para alcanzar la conformidad al refinar se añaden nodos (se dice entonces que el refinamiento se extiende por conformidad), parece lógico que, inversamente, al desrefinar se restrinja la zona desrefinada. Además, de esta manera, se asegura que los nodos eliminados verifican la condición de desrefinamiento.

Esto último no podría asegurarse si se lograra la conformidad eliminando aún más nodos de entre los que no cumplen la condición de desrefinamiento.

En cuanto a la conformidad local es bueno señalar que no se añade en la malla τ_j ningún nodo que no existiera previamente, como se hace en el algoritmo de refinamiento, sino que únicamente se cambian indicadores. Por último, si en algún nodo se cambia el indicador de "0" a "1" se hace necesario un nuevo bucle en elementos del nivel j-1 para asegurar la conformidad global.

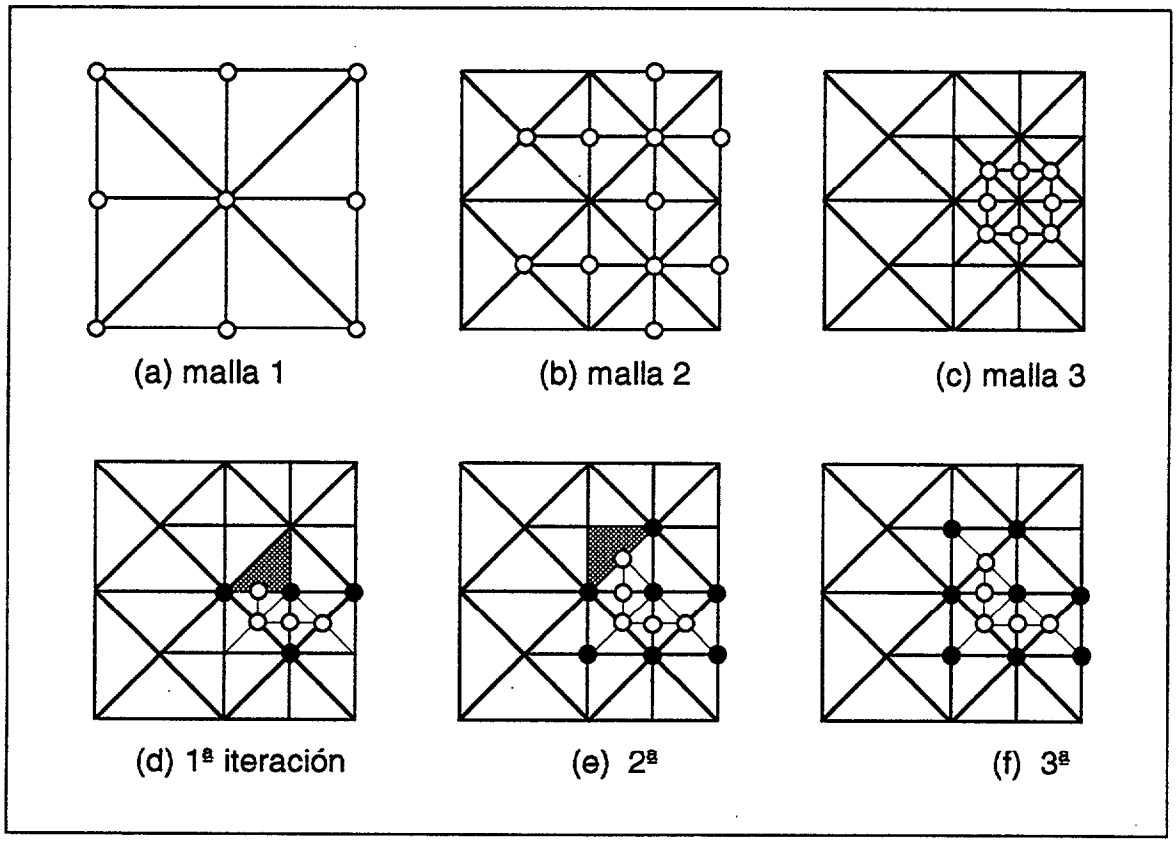


Figura 16.- La conformidad queda asegurada.
(a),(b) y (c) secuencia de mallas encajadas con nodos propios destacados.
(d), (e) y (f) nodos con indicador 1 de la 3ª malla (o) y anteriores (•).

Es claro que tanto el procedimiento de evaluación de la condición de desrefinamiento, como el de alcanzar la conformidad son finitos porque involucran un número de operaciones proporcional al número de nodos propios de cada nivel de malla. En la fig. 16 se muestra un ejemplo de cómo evoluciona la zona no conforme (en cuanto a los indicadores de

desrefinamiento de nodos) al pasar por Defcod, el tercer nivel de malla. La primera línea de la figura representa una secuencia de tres mallas encajadas. Supongamos que, de acuerdo con la condición de desrefinamiento, los nodos marcados en blanco en la fig. 16(d) deben permanecer. El triángulo sombreado es no-conforme. Por eso, algunos indicadores de desrefinamiento de algunos nodos deben cambiar. Las figuras 16(e) y 16(f) representan esos cambios. También se puede ver en la figura cómo se heredan los indicadores de nodos en cada bucle, a nodos de niveles anteriores (nodos marcados en negro) en cada iteración del "mientras".

Al final, la subrutina Defcod determina el tamaño en que se han de ampliar los vectores del saco para guardar en ellos los números de nodos, caras y elementos que se eliminarán, si es que se produce el desrefinamiento. Los nodos que se van a eliminar se han ido contando en los procedimientos que se detallan más arriba. En el caso de que el desrefinamiento sea *total*, es decir que todos los nodos propios de ese nivel de malla vayan a ser eliminados, el número de caras eliminadas es conocido, si sabemos el de nodos, pues responde a cualquiera de las dos fórmulas siguientes:

$$INCF = 2 \cdot INCN + NE(j) - NE(j-1) \qquad (1)$$

o bien:

$$INCF = NF(j) - NF(j-1) + INCN \qquad (2)$$

donde *INCF* es el tamaño en que se ampliará el vector del saco de caras e *INCN* el tamaño del de nodos. En este caso, es decir, suponiendo que el nivel τ_j desaparece al desrefinar, la expresión (1) puede escribirse también así:

$$NFP(j) = NFPE(j) + NFPI(j)$$

donde *NFP(j)* representa el número de caras propias de τ_j (que coincide con *INCF*), *NFPE(j)* es el número de caras externas de τ_j y *NFPI(j)* es el número de caras internas, pues se tiene:

$$NFPE(j) = 2 \cdot INCN = 2 \cdot NNP(j)$$
$$NFPI(j) = NE(j) - NE(j-1) = NEP(j)$$

Desgraciadamente, las fórmulas (1) y (2) sólo son válidas para el caso de que el desrefinamiento sea total. De todas formas, nos dan una cota superior del tamaño en que se deberá ampliar el vector de caras del saco, en cualquier desrefinamiento, parcial o total.

Respecto del número de elementos que se eliminarán, ni siquiera en el caso de desrefinamiento total creemos que existe una relación entre nodos, caras y elementos eliminados, pues este número depende además de la topología de la zona desrefinada, es decir del número de árboles que forman las caras que serán eliminadas utilizando una nomenclatura de teoría de grafos. De todas formas el vector del saco para elementos se amplía en una cota superior del espacio que se necesitará, en concreto en cuatro veces el espacio requerido para los nodos. Que este número no es excesivo lo muestra el hecho de que un único nodo introducido en el punto medio de una cara perteneciente a dos triángulos adyacentes provoca la aparición de cuatro nuevos elementos. Esto demuestra que, en el caso peor ésa es la cota superior mínima.

2.5.3. La ampliación de los vectores del saco

Los vectores del saco son ampliados por la subrutina Amem según el tamaño que ha determinado Defcod. Esta subrutina es la misma que se encarga de la ampliación del resto de vectores del código (véase el apartado 2.2.3.). Una vez que se haya realizado algún desrefinamiento, en sucesivos refinamientos el programa, antes de introducir un nodo, cara o elemento, asignará el primer número que se encuentre en el correspondiente vector del saco, empezando por el final. Por tanto esto implica que, si queremos combinar el desrefinamiento con el refinamiento, hay que hacer algunas pequeñas modificaciones en el proceso de refinamiento.

2.5.4. La redefinición de nuevos códigos: Subrutina Newind

El programa pasa por esta subrutina si en el nivel de malla en estudio no se elimina ningún nodo. El objetivo de esta subrutina es hacer unidad los indicadores de desrefinamiento de todos los elementos propios de este nivel de malla y también de todas las caras propias sin necesidad de pasar por la subrutina que definirá las nuevas conexiones nodales: Aquí no hay nuevas conexiones nodales que definir, puesto que no existe desrefinamiento.

La subrutina es necesaria para el correcto manejo de los indicadores. Los indicadores de los elementos no se han cambiado en la subrutina anterior, tras la evaluación de la condición de desrefinamiento, y han de ser actualizados a 1 los indicadores de los elementos que permanezcan en el mallado. También es necesario actualizar los indicadores de las caras porque la subrutina *Defcod* no asegura que *todas* las caras propias estén marcadas para la permanencia. Obsérvese que debido a la herencia de los indicadores de nodos a niveles de malla anteriores, puede ocurrir que ningún nodo propio de esta malla sea susceptible de ser eliminado y por tanto no sea preciso alcanzar la conformidad de la malla desrefinada. En ese caso tendríamos que las caras propias internas tendrán a la salida de *Defcod* indicador de desrefinamiento cero; este indicador debe cambiarse a 1 si no hay desrefinamiento.

Por último también se hacen igual a uno los indicadores de las caras padres y de los elementos padres de las caras y elementos, respectivamente, propios de este nivel, así como de los nodos extremos de las caras padres.

2.5.5. La compresión de mallas: Subrutina Congrid

En el caso de que se eliminen todos los nodos propios del nivel en curso, todo ese nivel desaparece de la secuencia de mallas. No se requiere, entonces, definir nuevas conexiones nodales sino eliminar de los vectores de estructura ese nivel. Esto lo realiza la subrutina Congrid.

Las tres partes de esta subrutina se refieren al manejo de los nodos, de las caras y de los elementos. En todas, en primer lugar, habrá que guardar los nodos, caras y elementos propios en el correspondiente vector del saco y, después, comprimir adecuadamente las colas de los vectores de estructura, excepto cuando el nivel de malla eliminado sea el último. Finalmente la subrutina cambia el tamaño de los vectores de estructura.

Supongamos que el nivel que desaparece es el nivel intermedio j, y que la secuencia de mallas consta de n niveles. Entonces:

a) En cuanto a los nodos:

Se recorren los nodos propios de τ_j. Para ello se sitúa un puntero en el lugar $NN(j-1)+1$ del vector de nodos *IMNODE*. Al mover el puntero desde $NN(j-1)+1$ hasta $NN(j)$, el correspondiente valor de *IMNODE* nos proporciona el número global de cada nodo propio de τ_j. Estos números globales se guardan en el vector *NNSAC*. Después hay que mover, hacia la izquierda, el resto del vector IMNODE, desde el lugar $NN(j+1)$ hasta el $NN(n)$ tantos lugares como nodos propios tenía el nivel que desaparece, es decir $NNP(j)=NN(j)-NN(j-1)$ lugares. A este respecto recuérdese que en el vector de estructura de nodos, sólo es preciso guardar los nodos propios de cada nivel (ver apartados 2.2.4. y 2.4.).

Como desaparece un nivel de malla, se ha de reasignar el nuevo número de nodos por nivel, que estamos llamando $NN(k)$, para k tal que $j \le k \le n-1$. Esta asignación de los nuevos números de nodos a los nuevos niveles de malla se hará mas adelante, cuando se manejen las colas de caras y elementos. Para ello se realizará un bucle en niveles de malla desde la siguiente a la que desaparece hasta la última.

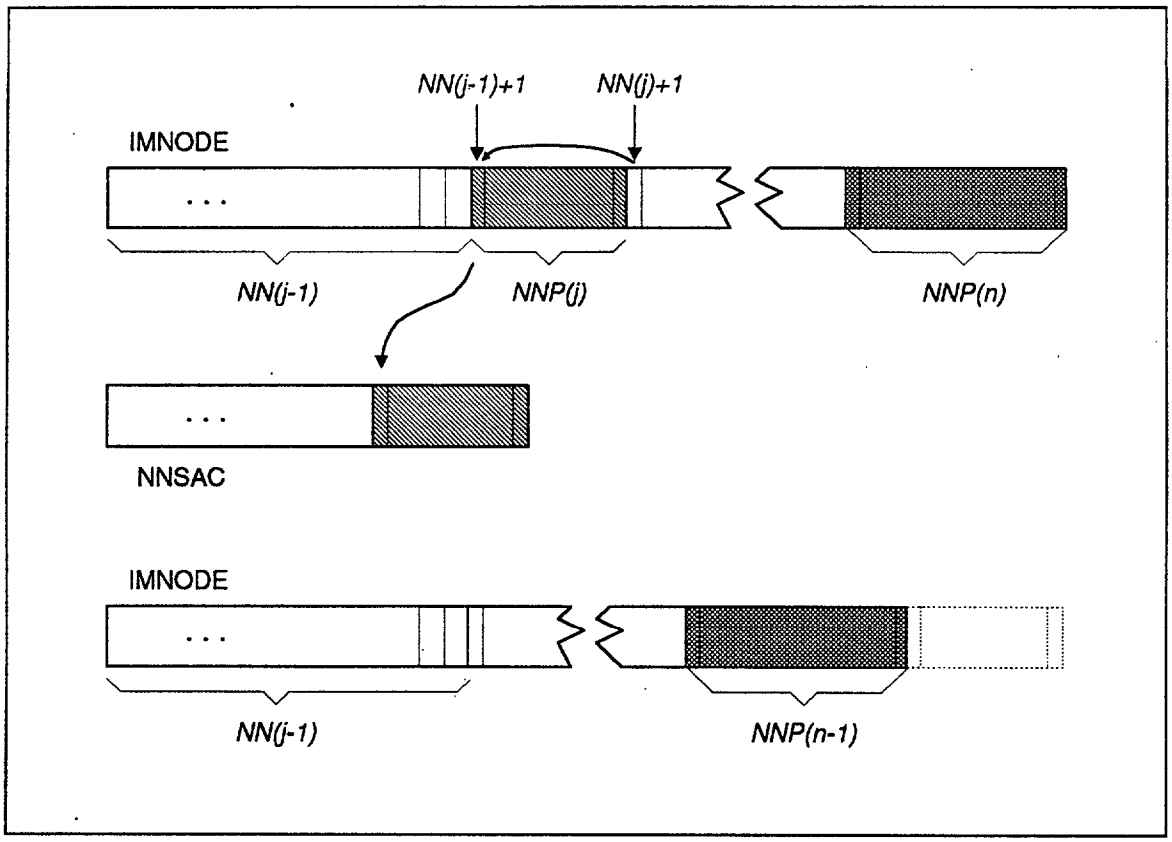


Figura 17.- Manejo del vector de nodos.

b) En cuanto a las caras:

Antes que nada hay que observar que todas las caras propias tendrán indicador de desrefinamiento *NFACES(·)*=0 y sólo ellas. Además se recuerda que en el vector de estructura de caras *IMFACE* se tienen todas las caras de cada nivel de malla, no sólo las propias como ocurría con los nodos. Por tanto para guardar en el saco de caras, *NFSAC*, las caras propias de τ_j lo único que hay que hacer es comprobar el valor del indicador de desrefinamiento para saber si tal cara es propia o heredada. Al mismo tiempo, y pensando en la posterior compresión de la "cola" de caras, se asigna *NFACES(NF)*=−1 a toda cara *NF* que sea cara propia de τ_j y primera cara hija, y se cuentan las caras propias de τ_j que no son primeras caras hijas. La razón de esta distinción entre caras propias que son primeras caras hijas y las que no lo son es la siguiente: toda cara propia de τ_j es cara heredada en los siguientes niveles de malla. Como toda cara propia de τ_j desaparece del mallado, en los

niveles siguientes al *j*-ésimo ha de ser sustituida por su cara padre, pero esta sustitución ha de afectar sólo a una de las dos caras que tienen la misma cara padre, pues cada cara aparece sólo una vez en el mallado. Naturalmente, las caras internas de τ_j no tienen cara padre, pero eso no ofrece ningún problema, porque desaparecerán también de los niveles siguientes.

Se cuentan las caras propias del nivel *j*-ésimo que no son primeras caras hijas porque en esta cantidad desciende el número total de caras en los niveles de malla posteriores al que se elimina, τ_j. Véase cómo se definen esos números en el *Procedimiento Comprime-caras I*, más adelante.

Sea el número de caras propias no primeras caras hijas *NFPN*. Para saber si una cara es primera cara hija o no, previamente a la hora de refinar habremos asignado a las primeras caras hijas $IR(3,NF)=-|IR(3,NF)|$, es decir hacemos negativo el signo de su cara padre.

Procedimiento Guarda-caras I:

ENTRADA: Vector de caras IMFACE, matriz de genealogía de caras IR, indicadores de desrefinamiento de caras NFACES, saco de caras NFSAC.

NFPN=0.
Para i=1, hasta NF(j), hace:
NC=i-ésima cara de τ_j.
Si NC es cara propia de τ_j , entonces:
- Guarda NC en el saco de caras NFSAC

Si es 1ª cara hija, entonces
- NFACES(NC)=-1

En otro caso
- Cuenta : NFPN=NFPN+1

Fin del si.
Fin del si.
Fin del bucle en caras.

SALIDA: IMFACE, NFACES actualizado, IR actualizado, NFPN y NFSAC actualizados.

Antes de detallar cómo se comprime la cola de caras, conviene resaltar el hecho de que si una cara puede ser eliminada, es porque no tiene caras hijas, propiedad análoga a la relativa a los elementos que ya se comentó. Como entonces, también ahora, esto quiere decir que si una cara puede ser eliminada, pertenece a la malla más fina de la secuencia de mallas encajadas existente en ese momento del proceso de desrefinamiento y por tanto a todos los niveles intermedios entre el que se elimina y el último. Esto explica que se tomen medidas en cuanto a la distinción de la primeras caras hijas de las que no lo son. Hay que pensar que al manejar los niveles de mallas posteriores al que desaparece, una de cada dos caras con la misma cara padre deberá ser sustituida por su cara padre en el vector de caras *IMFACE*. Además el número de caras propias de τ_j que no son primeras caras hijas es el número en que disminuirá el número de caras de todos los niveles posteriores al nivel que desaparece.

El manejo de la cola de caras se hace mediante un puntero y un contador de huecos. El puntero se sitúa en el lugar donde comienzan las caras del nivel j+1, sea éste *ISPF*. El número de huecos nos dice cuántos huecos hay que trasladar cada número de cara hacia la izquierda para comprimir el vector de estructura *IMFACE*. Al principio, es decir cuando *ISPF* está en la primera posición del sector del nivel j+1, este número de huecos que llamaremos *NHC* será igual al número de caras del nivel eliminado: *NHC=NF(j)*. Dentro de un bucle en niveles de malla desde el siguiente al que desaparece, j+1, hasta el último existente, n, *ISPF* recorre las caras de ese nivel; si una cierta cara debe permanecer su número ocupa el lugar dado por *ISPF–NHC*, si debe ser eliminada y no es primera cara hija se suma una unidad en el contador y, por último, si es primera cara hija, se guarda en el lugar que señala *ISPF–NHC* el número de su cara padre. Al finalizar el recorrido de las caras de cada nivel se asigna el nuevo número de caras al nivel anterior. Este número es el que teníamos menos el número de caras propias no primeras caras hijas del nivel que desaparece, j.

Este proceso se puede esquematizar como sigue:

Procedimiento Comprime-caras I:

ENTRADA: IMFACE, nº de caras de cada nivel NF(i) para i=j, ..., n, indicadores NFACES, matriz IR, NFPN.

NHC será el número de huecos en IMFACE. NHC=NF(j).

Para i=j+1 hasta n, hace:

Para k=1, hasta NF(i), hace:

KC=k-ésima cara de τ_i.

Si KC es cara propia de τ_j no primera cara hija,

- Aumenta el nº de huecos: NHC=NHC+1

En otro caso, si es cara propia de τ_j primera cara hija,

- Se asigna en el lugar correspondiente de IMFACE el nº de su cara padre.

En otro caso,

- Se cambia de lugar en IMFACE

Fin del si.

Fin del bucle en caras del nivel i-ésimo.

- NF(i-1)=NF(i)-NFPN

Fin del bucle en niveles de malla.

SALIDA: Vectores IMFACE, NF actualizados.

c) En cuanto a los elementos:

Los procedimientos que se siguen son similares a los de las caras. De nuevo los elementos propios del nivel *j* y sólo ellos tendrán indicador *NELES*=0. Como en los anteriores es necesario distinguir los elementos que son primeros elementos hijos, para que sean sustituidos por su elemento padre en los restantes niveles de malla. Nos limitaremos, pues, a esquematizar ambos procedimientos:

Procedimiento Guarda-elementos I:

ENTRADA: Vector de estructura IMELEM, matriz de genealogía de elementos IXH, indicador de desrefinamiento NELES, saco de elementos NESAC.

NEPN=nº de elementos propios de τ_j no primeros hijos.

NEPN=0.
Para i=1, hasta NE(j), hace:
IE=i-ésimo elemento de τ_j.
Si IE es elemento propio, entonces:
- Se guarda en el saco de elementos NESAC.
Si es primer elemento hijo, entonces:
- NELES(IE)=-1

Si no es primer elemento hijo:
- Se cuenta: NEPN=NEPN+1

Fin del si.
Fin del si.
Fin del bucle en elementos.

SALIDA: IMELEM, NELES actualizado, IXH, NEPN actualizado, NESAC actualizado.

Como antes también ahora el control de los elementos propios que son primeros elementos hijos es importante. En este caso, en el proceso de refinamiento, haremos *IXH(*6,·*)* negativo para tales elementos. Esto nos permite saber fácilmente si un elemento es primer hijo o no. La compresión del vector de elementos *IMELEM* y la adjudicación de nuevos números de elementos a los nuevos niveles restantes se hace como se explica a continuación.

Procedimiento Comprime-elementos I:

ENTRADA: IMELEM, nº de elementos por nivel NE(i) para i=j, ..., n, indicadores NELES, matriz IXH, nº de elementos propios no primeros hijos NEPN.

NHE=nº de huecos en IMELEM=NE(j).
Para i=j+1, hasta n, hace:
Para k=1, hasta NE(i), hace:
KE=k-ésimo elemento de τ_i.
Si es elemento propio de τ_j, entonces:
Si es primer hijo:
- Guarda en IMELEM, el nº de su padre.

Si no es primer hijo:

- Aumenta el hueco: NHE=NHE+1

Fin del si.

En caso contrario,

- Se cambia de lugar en IMELEM.

Fin del si.

Fin del bucle en elementos.

- NE(i-1)=NE(i)-NEPN.

Fin del bucle en niveles de malla.

SALIDA: Matriz IXH, vectores IMELEM y NE.

Se puede observar que el manejo de las colas de los vectores de estructura de caras y elementos *IMFACE* e *IMELEM*, se puede realizar a la vez, en el mismo bucle en niveles de malla. Desde luego esto sólo es necesario en caso de que el nivel que se elimina sea un nivel intermedio, es decir distinto del último.

Por último, la subrutina Congrid cambia el tamaño de los vectores *IMFACE* e *IMELEM*, liberando así parte de la memoria. El vector de nodos *IMNODE*, no lo cambia de tamaño porque tendrá siempre el tamaño del mayor número global de nodos que se encuentre o en la secuencia de mallas o en el saco. Sin embargo, aunque puede pensarse variar su tamaño como se hace con los otros vectores o eliminar el vector de nodos del saco, guardando estos nodos en el mismo vector *IMNODE* al final de él, al ser el vector más pequeño de los tres vectores de estructura, no se gana mucha memoria.

2.5.6. La definición de nuevas conexiones nodales: Subrutina Demesh

En caso de que el desrefinamiento del nivel de malla en curso, j, sea parcial, es necesario definir las conexiones nodales del nuevo nivel τ_j^j. También se habrán de actualizar los vectores de estructura y los de genealogía del nivel τ_j^j y de los siguientes: $\tau_{j+1}^j, \tau_{j+2}^j, \ldots, \tau_{n_j}^j$. Se da por tanto el caso más complicado. Como en la subrutina anterior, también en ésta deberemos manejar los nodos, las caras y los elementos.

a) En cuanto a los nodos:

Se recorren los nodos propios de τ_j. Los que deben ser eliminados se guardan en el saco de nodos *NNSAC*. El número de nodos eliminados es el número en que disminuirán los números de nodos de los niveles desde el j al último. A la vez que se guardan en el saco los nodos que se eliminan, se mueven en el vector *IMNODE* los nodos propios del nivel j hacia la izquierda, para lo cual se van contando los huecos que aparecen en este vector. Un esquema de lo anterior puede ser el que sigue:

Procedimiento Guarda-nodos:

ENTRADA: Vector IMNODE, nº de nodos propios del nivel j: NNP(j), NHN=número de huecos en IMNODE, indicador de nodos NODES, saco NNSAC.

NHN=0
Para i=1 hasta el nº de nodos propios de τ_j, hace:
  NI=i-ésimo nodo propio de τ_j.
  Si NI debe ser eliminado, entonces:
  - Se guarda en el saco de nodos NNSAC.
  - Cuenta huecos: NHN=NHN+1.

  Si NI debe permanecer:
  - Lo guarda en IMNODE, NHN lugares a la izquierda.

  Fin del si
Fin del bucle en nodos propios

- Asigna nuevo número de nodos al nivel j: NN(j)=NN(j)-NHN.

SALIDA: Vector IMNODE actualizado hasta el nivel j, nuevo nº de nodos del nivel j.

Como puede observarse en el procedimiento anterior, al mismo tiempo que se guardan en el saco los nodos que se eliminan, se actualiza parte del vector *IMNODE*. El manejo de la cola de nodos, es igual al que realizaba la subrutina Congrid. Lo único que hay que hacer es cambiar de posición en *IMNODE* todos los nodos propios de los niveles finales, desde $j+1$ hasta el último. En resumen este procedimiento guarda los números de los nodos eliminados en el saco de nodos y comprime el vector de nodos *IMNODE*. También se adjudican los nuevos números de nodos de esos niveles. Estos serán: *NN(i)=NN(i)-NHN*, para $i=j+1, \ldots, n$.

b) En cuanto a las caras:

El manejo de caras es bastante parecido a lo que hacía Congrid. Ahora en el nivel de malla j tendremos caras marcadas con "2" (las caras heredadas de niveles anteriores), caras marcadas con "1" (las caras propias que deben permanecer) y caras marcadas con "0" (las caras propias que en principio pueden ser eliminadas).

Sin embargo hay que hacer notar que la subrutina Defcod sólo cambia indicadores de desrefinamiento a las caras propias externas, así que entre las caras marcadas con "0", tendremos algunas caras internas que deban permanecer. Por tanto si una cara está marcada con "0" habremos de comprobar si sus dos nodos extremos deben permanecer, ya que si esto es así, también esa cara debe permanecer en la malla. Basta con que desaparezca uno de los nodos extremos para que la cara correspondiente deba ser eliminada; esto es, precisamente lo que ocurre con las caras internas.

Por ejemplo, en la fig. 12 (pág. 35), si al desrefinar la malla τ_j, el nodo N_2 se puede quitar y el nodo N_1 no, la cara interna que los une debe ser eliminada.

Como en la Subrutina Congrid también asignaremos indicador "-1" a las caras propias que sean primeras caras hijas para la posterior compresión del vector *IMFACE*.

Procedimiento Guarda-caras II:

ENTRADA: Vector de caras IMFACE, matriz de genealogía IR, indicadores NFACES, saco de caras NFSAC, nodos extremos de las caras: matriz IC, número de caras de τ_j= NF(j).

NHC=número de huecos en IMFACE=0.
Para i=1, hasta NF(j), hace:
NC=i-ésima cara de τ_j, N1, N2 nodos extremos de NC.
Si NFACES(NC)=0, entonces:
Si N1 ó N2 se eliminan, entonces:
- Guarda NC en el saco.

Si es 1ª cara hija, entonces:
- NFACES(NC)=-1.
- Guarda en IMFACE el nº de su padre.

En otro caso:
- Aumenta el hueco: NHC=NHC +1.

Fin del si.
En otro caso, si N1 y N2 permanecen, entonces:
- NFACES(NC)=1

Fin del si.
En otro caso:
- Guarda NC en IMFACE.

Fin del si.
Fin del bucle en caras.
- Actualiza nuevo nº de caras: NF(j)=NF(j)-NHC

SALIDA: IMFACE y NFSAC actualizados, número de huecos NHC, indicadores NFACES actualizados, nuevo número de caras de τ_j= NF(j).

El procedimiento para comprimir la cola del vector *IMFACE*, es muy parecido al empleado en Congrid, salvo que ahora no perdemos niveles de malla y el tamaño en que disminuyen los números de caras no será, evidentemente, el número de caras propias no primeras caras hijas de τ_j. Este procedimiento se puede esquematizar así:

Procedimiento Comprime-caras II:

ENTRADA: IMFACE, nº de caras de cada nivel de malla, NFACES, matriz IR, NHC.

- Guarda NHC en NHCI: NHCI=NHC.

  Para i=j+1, hasta n, hace:

    Para k=1, hasta NF(i), hace:

      KC=k-ésima cara de τ_i.

      Si NFACES(KC)=0, entonces:

      - Aumenta el hueco: NHC=NHC+1

      En otro caso, si NFACES(KC)=-1:

      - Se asigna en el lugar trasladado de IMFACE el nº de su cara padre.

      En otro caso (NFACES(NC)>0):

      - Se cambia de lugar NC en IMFACE

      Fin del si.

    Fin del bucle en caras.

  - Asigna nuevo nº de caras al nivel: NF(i)=NF(i)-NHCI.

  Fin del bucle en niveles de malla.

  Se disminuye el tamaño de IMFACE:

- IM(4,IMFACE)=IM(4,IMFACE)-NHC.

SALIDA: IMFACE actualizado, nuevo nº de caras de cada nivel, matriz IR.

c) En cuanto a los elementos:

Como estamos en el caso de desrefinamiento parcial, podrán aparecer ahora elementos que antes no existían. Esto requerirá un control de la información (indicadores de desrefinamiento) así como distinguir cada una de las posibilidades que se dan tanto al refinar como al desrefinar un elemento. Se utilizarán aquí dos punteros: *ISPE* e *ISPE1*. El primero recorrerá los elementos del nivel de malla anterior al que estamos desrefinando: τ_{j-1}, el segundo servirá para re-escribir en *IMELEM*, los nuevos elementos de la malla τ_j^j, es decir, al principio estará situado en el primer lugar de la zona de *IMELEM* del nivel *j*.

Antes de dar unos esquemas de los procedimientos correspondientes a los elementos, una observación: en la entrada de esta subrutina, los elementos de τ_{j-1} pueden tener los siguientes indicadores:

NELES(NE)=2 si *NE* es elemento de τ_{j-1} que puede ser heredado en la malla τ_j o con hijos en τ_j.

NELES(NE)=1 si *NE* es un elemento de τ_{j-1} que también pertenece a τ_j y con hijos en algún nivel posterior a τ_j.

En cuanto a la primera posibilidad es claro si se piensa cómo actuaba la subrutina Indesr (pág. 44) y cómo reinicializa los indicadores la subrutina Defcod (pág. 45). Además los indicadores de elementos que permanecen (*NELES*=1) se adjudican también al elemento padre sólo en la subrutina Newind y en ésta, Demesh. Por tanto, si en la iteración anterior del bucle en niveles de malla, es decir al desrefinar el nivel $j+1$, se heredó al padre de algún elemento el indicador *NELES*=1, éste sólo afectaría al nivel de malla $j-1$ si ese elemento era heredado en τ_j con hijos en τ_{j+1}. Así que, si algún elemento de τ_{j-1} tiene indicador 1, entonces debe estar también en el nivel j y además tiene sucesores en niveles posteriores.

Respecto del nivel de malla τ_j los posibles indicadores de elementos y significados son:

NELES(NE)=1 si *NE* debe permanecer porque tiene algún descendiente.

NELES(NE)=0 si *NE* es elemento propio sin descendientes. Puede desaparecer.

NELES(NE)=2 si *NE* es un elemento heredado que no tiene sucesores, es decir que puede desaparecer, pero no en esta iteración del bucle, sino al desrefinar el nivel en que es propio.

Procedimiento Guarda-elementos II:

ENTRADA: Vector de estructura de elementos IMELEM, matrices de genealogía de caras IR y de elementos IXH, indicador de elementos NELES, saco de elementos NESAC, matriz IC de nodos por cara, matriz IX de caras por elemento.

NHE=número de huecos en IMELEM=0.
Para i=1, hasta NE(j-1), hace
NE=i-ésimo elemento de τ_{j-1}.
Si NELES(NE)=2, entonces
- Cuenta los hijos que le quedarán.

Si tenía hijos, dependiendo de los hijos que tenía,
- Define conexiones de los nuevos hijos.
- Almacena en el nivel j de IMELEM los nuevos hijos.
- Guarda en NESAC lo números de los que se quitan.
- Cuenta los huecos que quedan en IMELEM.

En caso contrario:
- Guarda en el nivel j de IMELEM, NE.

Fin del si.
En caso contrario, si NELES(NE)=1, entonces:
- Guarda en el nivel j de IMELEM, NE.

Fin del si.
Fin del bucle en elementos de τ_{j-1}.
- Asigna nuevo nº de elementos de τ_j.

SALIDA: IMELEM actualizado, IXH actualizada, NHE actualizado, NELES actualizado, nuevo NE(j), matrices IR, IC, IX.

No se ha detallado en el esquema anterior, la forma en que se definen en cada caso las nuevas conexiones nodales, aunque requiere distinguir todos los casos posibles. En la fig. 18 se representa el caso en el que un elemento con 4 hijos pasa a tener tres al desrefinar; el resultado es un elemento refinado a siniestra. La flechas sobre las caras indican sentido de recorrido interno en la cara. Las flechas curvas en los elementos significan numeración local de las caras para tal elemento, comenzando desde donde parte la flecha. Por último los números en un círculo quieren decir

numeración local de los hijos con respecto al elemento padre, es decir primer elemento hijo, segundo, etc. Toda esta información está disponible a la entrada de la subrutina y manejada adecuadamente nos permite redefinir las nuevas conexiones nodales. Se ha de resaltar aquí que toda la información que está representada en la figura, corresponde exactamente a dos de las posibilidades en que puede dividirse un elemento (véase fig. 6 en la pág. 23) y manifiesta la dificultad de programación de esta subrutina. Como se aprecia en la figura, pueden aparecer elementos que no existían en la secuencia de entrada, como el tercer hijo en el triángulo de la derecha. Estos elementos los llamamos *semi-hijos* porque siempre son resultado de la bisección de su elemento padre.

Se puede resaltar que la redefinición de las conexiones nodales de los elementos ha de estar de acuerdo con el proceso de caras que se explicó con anterioridad. Por otro lado, de forma semejante a como se hizo en la subrutina Congrid, a los elementos primeros elementos hijos les asignamos *NELES(·)*=-1, *cuando un elemento pierde todos sus hijos*, para el posterior manejo de la cola de elementos. Además cuando un elemento sigue teniendo algún elemento hijo, éstos toman indicador 1 y su padre también.

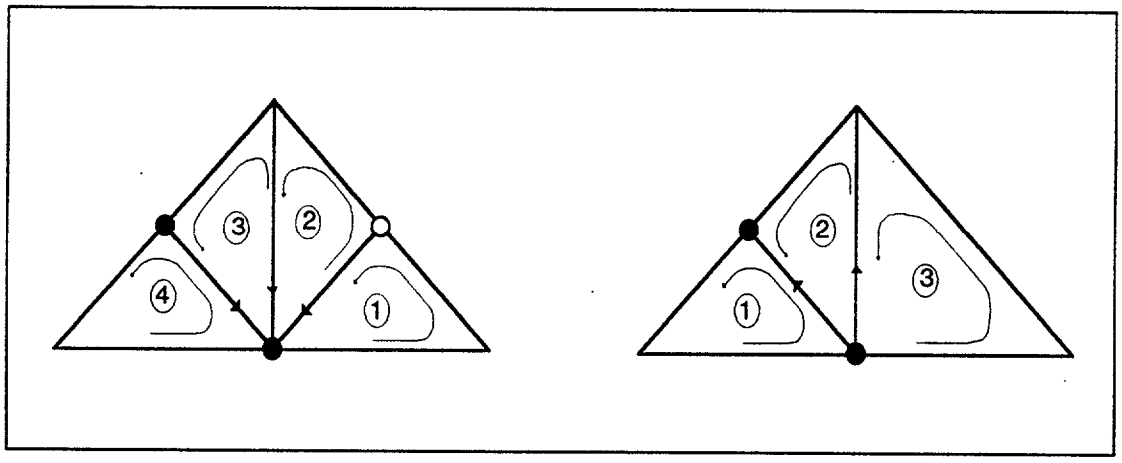


Figura 18.- Ejemplo de redefinición de conexiones nodales.

Después de la definición del nuevo nivel de malla *j*, estos cambios han de ser trasladados al resto de la estructura de mallas. Este proceso es semejante al que se siguió en Congrid. Aquí se puede hacer notar que basta con recolocar en *IMELEM* los números de elementos que quedan, pues estos

números son algunos de los que existían antes en ese nivel, aunque, quizá hayan cambiado sus conexiones nodales. Así pues, el procedimiento que sigue al final de esta subrutina es el mismo que se tenían en Congrid, lo llamábamos *Comprime-elementos I.* También, como en Congrid, al final, se cambia el tamaño de los vectores *IMFACE* e *IMELEM.*

2.6. Observaciones

Sobre el algoritmo de desrefinamiento que se acaba de describir se pueden resaltar las siguientes observaciones o características:

2.6.1. Número de secuencias intermedias

Al final de cada iteración, del bucle en mallas, resulta una nueva estructura de mallas encajadas menos fina que la anterior. De esta manera, se obtienen tantas secuencias como números de niveles de mallas menos uno. Esto se puede representar así: $\mathbf{T} \geq \mathbf{T}^{n} \geq \mathbf{T}^{n-1} \geq \ldots \geq \mathbf{T}^{2} = \mathbf{T}^{'}$. El superíndice indica el número de nivel recientemente desrefinado en esa iteración. Lógicamente, como se ha visto, de una secuencia a otra cambia en principio toda la subsecuencia desde el nivel recién desrefinado al último existente en ese momento. Además el número de niveles puede disminuir si algún nivel desaparece por completo. Eso se indicó en el primer esquema del algoritmo con el subíndice n_j. Es decir, si n_j representa el número de niveles de malla que quedan tras desrefinar el nivel j, se tiene la siguiente relación entre el número de niveles correspondientes a dos secuencias sucesivas:

$$n_j = n_{j+1} - 1 \quad \text{ó} \quad n_j = n_{j+1}$$

No es necesario que el bucle en niveles de malla termine en el segundo nivel, también se puede realizar desde el último hasta un nivel de malla que entraría como dato. Esto podría ser particularmente útil si se partió de una malla inicial muy grosera y equivaldría a tomar como malla inicial, no la que definía las especificaciones geométricas del dominio sino otra más fina que capte, por ejemplo, las condiciones iniciales del problema.

La forma que en el algoritmo se van obteniendo estos diversos niveles de malla intermedios queda reflejado en la fig. 19. En ella se ha realizado el desrefinamiento hasta el segundo nivel de malla. Tanto en la primera línea, secuencia original, como en la última, secuencia final, se han resaltado los nodos propios de cada nivel. La secuencia original es la que aparecía en la fig. 16 (pág. 51) con un nivel más de división. La malla inicial se ha omitido

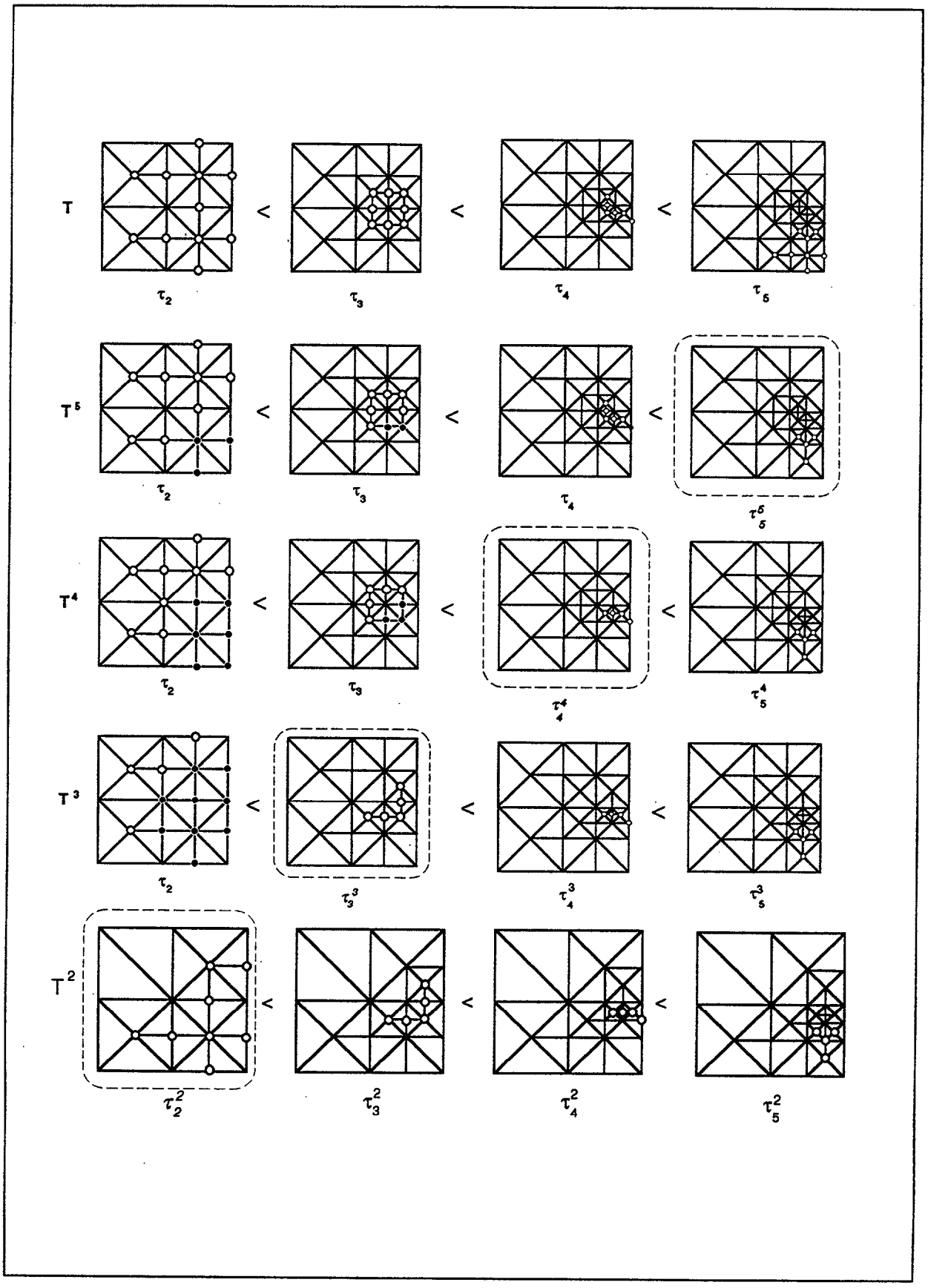


Figura 19.- Las secuencias intermedias del algoritmo de Desrefinamiento.

porque permanece inalterable a lo largo de todo el proceso. El nivel resaltado con puntos es el desrefinado en cada iteración del bucle en niveles de malla. En las secuencias intermedias T^l se ha reflejado cómo se heredan los indicadores de nodos $NODES(\cdot)=1$ a niveles de mallas anteriores (en negro) y también se señalan los nodos propios de cada nivel susceptibles de ser eliminados (en blanco).

Obsérvese que cuanto mayor es el número de mallas encajadas más son los nodos que deben permanecer si permanece uno de los nodos de malla más fina de la secuencia. Así, en el ejemplo de la figura, al desrefinar el segundo nivel de malla, sólo hay cuatro nodos susceptibles de ser eliminados, mientras que otros nueve deben permanecer. En la figura no disminuye el número de niveles porque, en cada nivel, permanece algún nodo propio.

Véase también, cómo se heredan los cambios a las subsecuencias finales.

Por otra parte el que aparezcan tantas secuencias de mallas dentro del algoritmo de desrefinamiento no tiene demasiada importancia, ya que cada una de estas secuencias desaparece cuando es creada la siguiente.

2.6.2. El problema algebraico: la renumeración de las ecuaciones

En el apartado anterior 2.5 se ha expuesto el problema geométrico del desrefinamiento, pero, como ya se indicó, el desrefinamiento geométrico lleva asociado un problema algebraico que consiste en la renumeración de las ecuaciones asociadas a cada grado de libertad por nodo. Es decir, tras desrefinar, hay que volver a asignar un número de ecuación a cada grado de libertad porque, lógicamente, al quitar nodos, perdemos también ecuaciones del sistema algebraico asociado al método de los elementos finitos.

Ante este problema hemos optado por recalcular qué número de ecuación corresponde a cada grado de libertad y asignar nuevos números de ecuaciones a cada nivel de malla. Además, con objeto de guardar la solución correspondiente a cada número de ecuación habrá que adjudicar la solución del antiguo número de ecuación al nuevo número de ecuación. Se consigue de esta forma que la solución numérica asociada a la secuencia de mallas desrefinada sea la mejor aproximación, en el sentido de interpolación, de la solución en la secuencia previa al desrefinamiento. De todo ello se encarga la Subrutina Dreneq; ésta actúa tras el desrefinamiento geométrico. Después, se podrá resolver el sistema de ecuaciones por el método que se haya elegido. Como ya se ha dicho, el código Neptuno es capaz de utilizar un método Multimalla combinado con el refinamiento y el desrefinamiento en mallados estructurados.

Seguidamente se explicará cómo trabaja la Subrutina Dreneq. Un esquema puede ser:

ENTRADA: Vectores IMNODE e IMFACE, nº de grados de libertad por nodo NDF, número de ecuaciones de la secuencia antes de ser desrefinada NEQ, números de referencia de los nodos NUREFN, tabla IREF, vector de número de ecuación por grado de libertad por nodo ID, números de nodos y ecuaciones de cada nivel de malla: NN(j), NQ(j), con j=1,...,n, solución por cada número de ecuación U(1:NEQ); contador NQQ.

NQQ=0.

Para KM=1, hasta n, hace:

NNP(KM) = nº de nodos propios del nivel KM.

Para i=1, hasta NNP(KM), hace:
Ni = i-ésimo nodo propio del nivel KM.
NR=número de referencia de Ni.
Si NR=0, entonces
Para j=1, hasta NDF, hace:
- Cuenta el nº de ecuación: NQQ=NQQ+1
- Adjudica la antigua solución al nuevo número de ecuación.
- Asigna el nuevo nº de ecuación en la tabla ID: ID(j,Ni)=NQQ

Fin del bucle en grados de libertad.
En otro caso:
Para j=1, hasta NDF, hace:
Si IREF(j,NR) ≤ 0, entonces:
- Cuenta el nº de ecuación: NQQ=NQQ+1
- Adjudica la antigua solución al nuevo nº de ecuación.
- Asigna el nuevo nº de ecuación en la tabla ID: ID(j,Ni)=NQQ.

Fin del si.
Fin del bucle en grados de libertad.
Fin del si.
- Adjudica nuevo nº de ecuaciones al nivel KM: NQ(KM)=NQQ.

Fin del bucle en niveles de malla.
- Cambia el tamaño de los vectores dependientes del número de ecuaciones.

SALIDA: Nueva matriz ID, vector solución U, nº total de ecuaciones NEQ, nuevos números de ecuaciones por nivel de malla NQ(j), j=1,..., n.

Conviene señalar que tanto en la adjudicación de la antigua solución al nuevo número de ecuación como en la asignación del nuevo número de ecuación al correspondiente grado de libertad por nodo interviene la tabla *ID*: la antigua *ID* para averiguar la antigua solución para tal grado de libertad por nodo, y la nueva *ID* que es definida en esta subrutina. Como se ve al mismo

tiempo que se define la nueva tabla *ID*, se utiliza la antigua. Esto es posible hacerlo porque en esta readjudicación de números de ecuación por grado de libertad por nodo, *siempre se asigna un número de ecuación menor o igual al que tenía ese grado de libertad por nodo previamente al desrefinamiento.* Así que la redefinición de la tabla *ID*, va *por detrás* de su utilización para conocer la antigua solución.

Se puede recordar aquí también, que la tabla *IREF* nos dice para cada número de referencia, el tipo de especificación que éste representa y que sólo $IREF(j,NR) > 0$ significa condición de tipo Dirichlet en el grado de libertad j del número de referencia *NR*, por tanto, en cualquier otro caso, es decir si $IREF(j,NR) \leq 0$ tendremos que asignar un número de ecuación.

Por último, se puede observar que los números de ecuación asignados son consecutivos, así que no hay que cambiar el método de resolución que tenga incorporado el código.

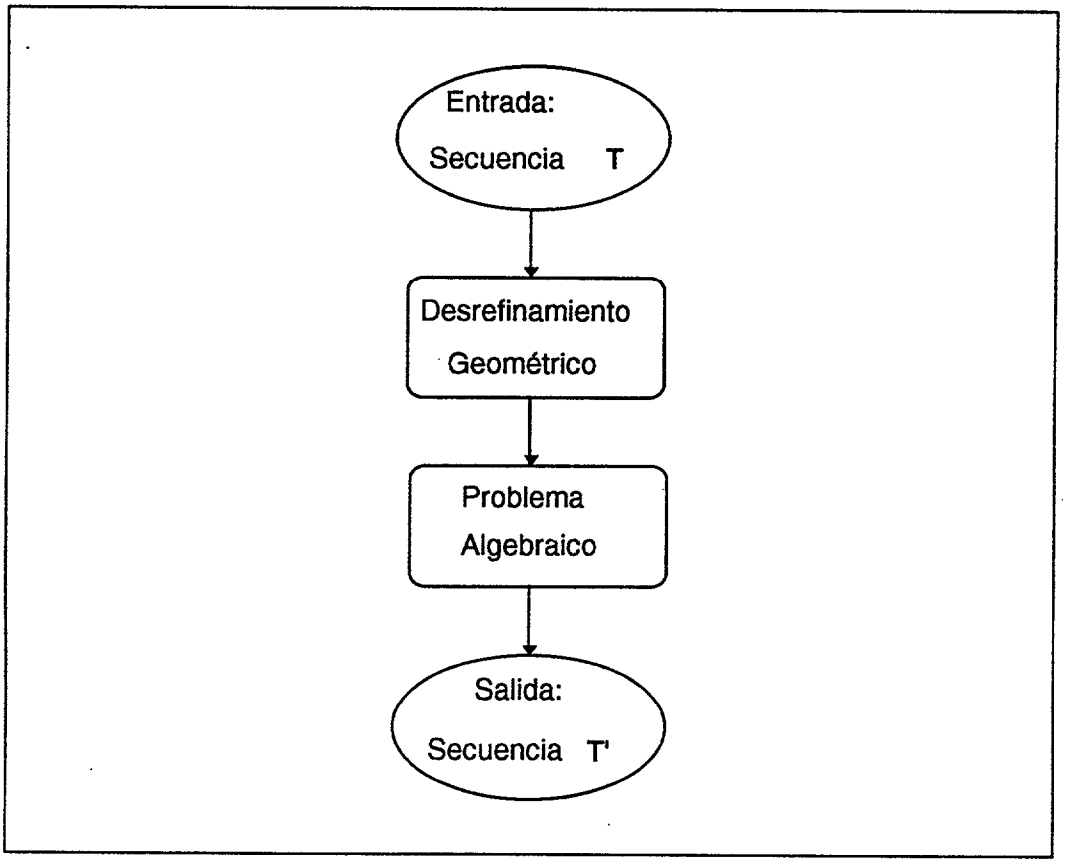


Figura 20.- Las dos partes del Algoritmo de Desrefinamiento.

2.6.3. La compactación de niveles

Tal como se ha descrito el algoritmo de desrefinamiento hasta el momento, se tiene que la permanencia de un sólo nodo por no cumplir la condición de desrefinamiento, hace que todo ese nivel de malla permanezca en la secuencia desrefinada. Como puede verse, esto hace que, dependiendo del parámetro épsilon que regula la condición de desrefinamiento, la aplicación reiterada de refinamientos y desrefinamientos tienda a generar secuencias de mallas encajadas, en las que mallas consecutivas difieren en un número muy pequeño de nodos. Este hecho es un serio inconveniente en cuanto a las buenas propiedades de aplicación del método multimalla, en tales secuencias.

Por otra parte, como cada vez que se realiza un refinamiento (local) se aumenta en uno el número de niveles de mallas tras varios refinamientos y desrefinamientos, se ha comprobado que el número de niveles de mallas tienen a crecer, mientras que el número de nodos tiende a permanecer constante. Así, se pueden tener secuencias en que la ratio números de nodos por número de niveles sea de unas pocas unidades.

Respecto del número creciente de niveles de malla que se generan se puede recalcar que estos niveles no corresponden al mismo grado de división ó profundidad del mallado inicial. Es decir, si tras varios refinamientos y desrefinamientos se tienen, por ejemplo, 50 niveles de malla distintos y sólo mil nodos, esto no quiere decir que la malla más fina requiera 50 niveles de división mediante el algoritmo 2-T o el 4-T de Rivara. Desde luego se puede obtener esa misma secuencia de 50 niveles partiendo de la misma malla inicial y utilizando el algoritmo 2-T (ó el 4-T) de Rivara 50 veces, sin embargo, quizá sean suficientes menos niveles de subdivisión. Este problema se ha resuelto, en parte, mediante un algoritmo que llamaremos de *compactación de niveles*.

Para explicar mejor este algoritmo, pensemos un ejemplo sencillo:

Supongamos que en un determinado momento de un proceso que llamamos readaptativo (combinación de refinamientos y desrefinamientos), se tienen 4 niveles de malla, de tal manera que todo elemento propio del 4º nivel es hijo de algún elemento propio del segundo. Es claro entonces que todo

elemento propio del tercer nivel es heredado en el cuarto. Por tanto, el tercer nivel puede ser eliminado de los vectores de estructura de caras y elementos. Ese nivel puede desaparecer.

> *La condición necesaria y suficiente para que en una secuencia de mallas encajadas distintas, un determinado nivel pueda ser eliminado, es que sus elementos propios no tengan hijos en el siguiente nivel.*

Un ejemplo de una secuencia en la que se puede comprimir un nivel de malla se muestra en la fig. 21. En ella aparece una secuencia de cuatro niveles de malla en la que un nivel puede desaparecer.

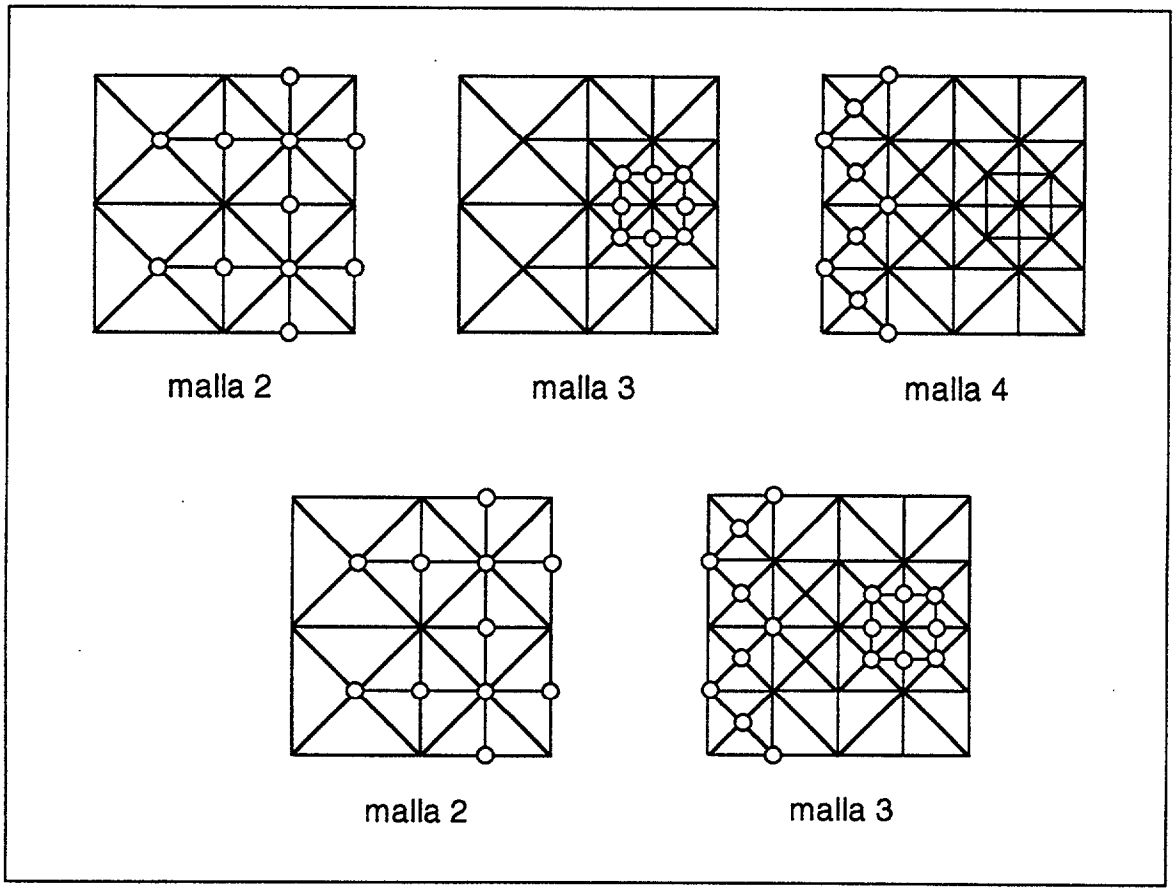


Figura 21.- La compactación de niveles.
En la 1ª línea el tercer nivel puede desaparecer.

El diagrama de flujo del algoritmo de compactación de niveles queda reflejado en la figura siguiente.

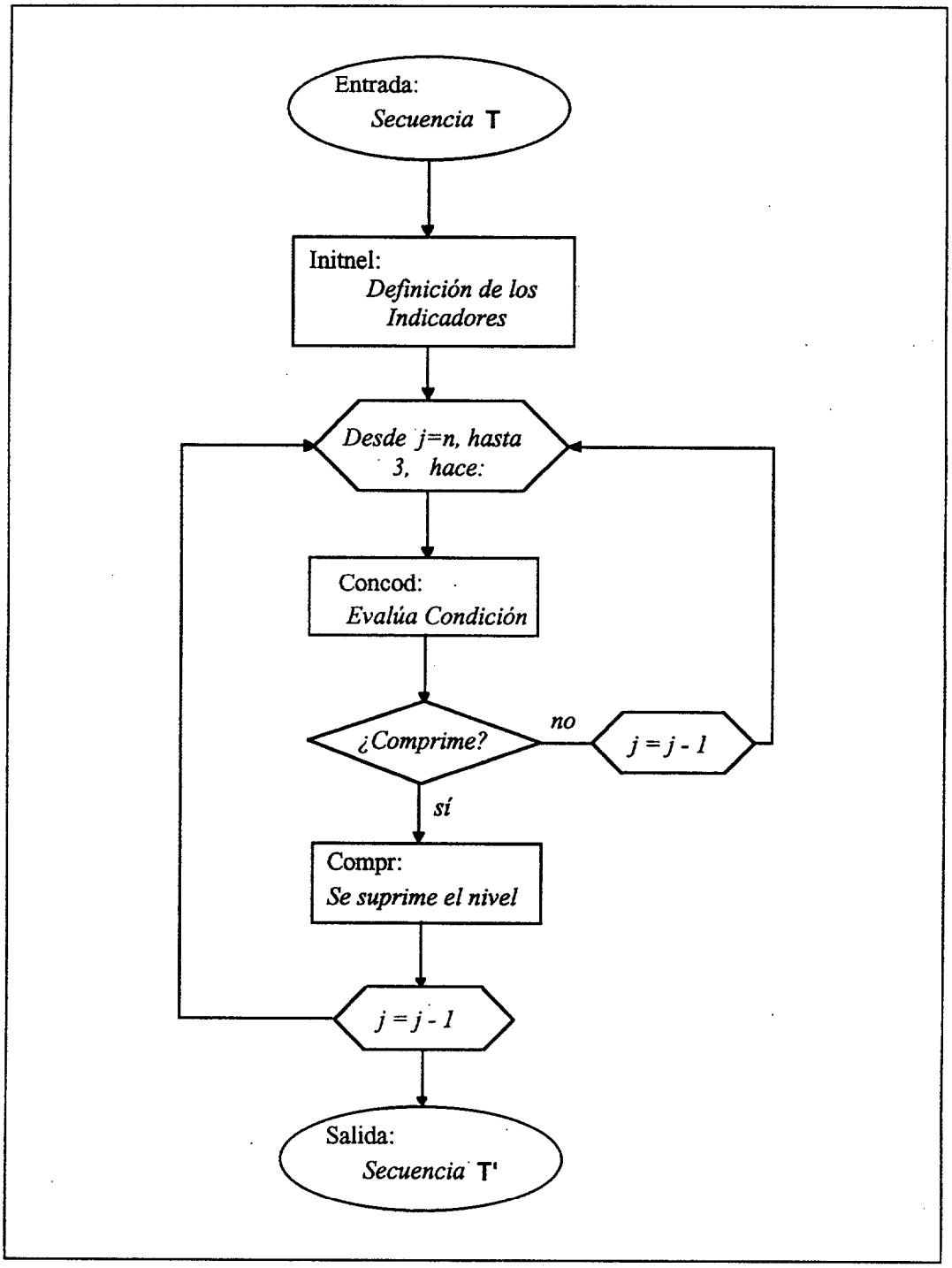


Figura 22.- Diagrama de flujo del algoritmo de compactación de niveles.

En la figura anterior **T'** denota la secuencia obtenida tras la compactación de niveles. No es por tanto la secuencia resultado de un desrefinamiento, aunque las salidas de ambos procesos se han significado de la misma forma.

A continuación se comentan las subrutinas que componen este algoritmo.

La definición de indicadores. Subrutina Initnel.

Se usan, en el algoritmo que estamos viendo, como indicadores de si existirá compresión de un determinado nivel de malla o no los mismos indicadores de desrefinamiento de los elementos. Sin embargo deben ser definidos de otra forma. Esto lo realiza la Subrutina Initnel, que puede esquematizarse así:

ENTRADA: Vector IMELEM, vector de indicadores NELES, NE(j) para j=1,..., n.

Para j=n, hasta 1, hace:

Para i=1, hasta NE(j), hace:

Ni=i-ésimo elemento de la malla τ_j.

- NELES(Ni)=j

Fin del bucle en elementos

Fin del bucle en niveles de malla

SALIDA: Vector NELES redefinido.

Como puede verse, lo único que hace esta subrutina es asignar a cada elemento el número del nivel en que es propio, es decir, el número del nivel de malla en que fue creado. Está claro que ese número es único para cada secuencia de mallas encajadas y por tanto este indicador *NELES* que actúa ahora como indicador de nivel está bien definido.

La evaluación de la condición: Subrutina Concod.

Esta subrutina evalúa si el nivel de malla anterior al que le llega puede desaparecer o no. Por este motivo el bucle en niveles de mallas de que consta el algoritmo de compactación va desde la última malla (nivel n) hasta la tercera – véase fig. 22 –. Sólo puede desaparecer un nivel que esté comprendido entre el segundo y el n–1.

Esta Subrutina devuelve una variable lógica que nos indica si hay o no *compactación* del nivel de malla inmediatamente *anterior* al que le llega.

ENTRADA: Vector IMELEM, matriz de genealogía de elementos IXH, indicadores de nivel NELES, nivel de malla j, variable lógica LCOM.

LCOM=verdadero

Para i=1, hasta NE(j), hace:

Ni=i-ésimo elementos de τ_j.

NPi=elemento padre de Ni

Si NELES(NPi) > j – 2, entonces:

- No hay compactación: LCOM=falso

Fin del si.

Fin del bucle en elementos.

SALIDA: Indicador de compactación LCOM.

Obsérvese que todo elemento de τ_j tiene indicador de nivel menor o igual que j; tendrá indicador j si es elemento propio de ese nivel y menor que él si es heredado. Más arriba se señaló que un nivel de malla $j-1$ podrá ser eliminado si y sólo si sus elementos hijos no están en el siguiente nivel. En esta subrutina en lugar de comprobar esta condición se estudia la condición equivalente siguiente: si todos los elementos del nivel j tienen sus padres en niveles anteriores o iguales al $j-2$.

Que ambas condiciones son equivalentes es evidente, pues si ningún elemento propio del nivel $j-1$ tiene sus hijos en el siguiente nivel, todos ellos serán heredados en el nivel j. Así que todo elemento de τ_j tendrá su elemento padre como mucho en nivel $j-2$. Recíprocamente, es también trivial: si todo elemento de τ_j tiene su padre a lo más en el nivel $j-2$, los elementos propios de τ_j no serán hijos de ningún elemento propio de τ_{j-1} así que se cumple la primera condición.

Por último, como es suficiente que un elemento no cumpla la condición para que la compactación del nivel no sea posible hacerla, se puede finalizar la subrutina tras la primera vez que se hace la variable lógica *LCOM* falso.

La compactación de un nivel: Subrutina Compr.

Supongamos que el nivel $j-1$ puede ser eliminado. Esta eliminación consistirá en la readjudicación de números de nodos, caras y elementos a los nuevos niveles de malla desde el $j-1$ al $n-1$. Además habrá que suprimir de los vectores de estructura de caras y elementos los respectivos tramos correspondientes al antiguo nivel $j-1$. Todo ello lo realiza la Subrutina Compr que se puede resumir así:

ENTRADA: Vectores IMFACE, IMELEM, nivel de malla que desaparece: j-1.

Para KM= j – 1, hasta n – 1, hace:

Para i= 1, hasta NE(KM+1), hace:

- Mueve cada nº, NE(j – 1) lugares a la izquierda.

Fin del bucle en elementos

- Asigna nuevo nº de elementos: NE(KM)=NE(KM+1)

Para i=1, hasta NF(KM+1), hace:

- Mueve cada nº, NF(j – 1) lugares a la izquierda.

Fin del bucle en caras

- Asigna nuevo nº de caras: NF(KM)=NF(KM+1)
- Asigna nuevo nº de ecuaciones: NQ(KM)=NQ(KM+1)
- Asigna nuevo nº de nodos: NE(KM)=NE(KM+1)

Fin del bucle en niveles de malla.

- Cambia el número de niveles: n = n – 1.
- Reduce tamaño de IMFACE e IMELEM.

SALIDA: Nuevos vectores IMFACE e IMELEM, nuevo número de niveles de malla.

Señalamos a continuación algunas características de este algoritmo de *compactación* de niveles.

Propiedades.

1) En primer lugar se puede señalar que de acuerdo con la definición de secuencia más fina que otra que se introdujo al principio del apartado 2.5.

(pág. 41), la secuencia final del algoritmo de compactación T' es menos fina que la secuencia de entrada T.

Además también se tiene que $T \leq T'$. Sin embargo ambas secuencias no son, en general, iguales. Sólo serán iguales en el caso de que no se haya compactado ningún nivel de malla. De acuerdo con estas definiciones se puede decir que son secuencias equivalentes.

2) La eficacia del algoritmo ha quedado suficientemente demostrada en numerosos ejemplos. Naturalmente es más eficaz en problemas con regiones de refinamiento móviles, por ejemplo, problemas de convección-difusión; y más en aquellos con término de convección predominante. El número total de niveles eliminados por la compactación de mallas puede llegar a centenares, con lo que esto supone de ahorro de memoria.

3) Hay que distinguir este algoritmo del de desrefinamiento. En éste se pierden niveles de malla para obtener otra secuencia equivalente a la original, pero no se elimina ningún nodo, sólo se cambian de nivel. En el algoritmo de desrefinamiento el objetivo es eliminar los nodos superfluos aunque, circunstancialmente, también se eliminen niveles de malla, precisamente cuando desaparecen todos los nodos propios de ese nivel. De forma que en el algoritmo de desrefinamiento se obtiene en general una secuencia menos fina que la de entrada. Sólo se obtiene la misma secuencia si no ha habido desrefinamiento.

4) Aunque no habría inconveniente en incorporar este algoritmo como un módulo independiente en el código, de forma que el usuario eligiera cuándo realizar una compactación, parece mejor que esta compactación se realice de forma automática. De hecho se realiza la compactación de mallas siempre que se desrefina, de manera que podríamos decir que el algoritmo de desrefinamiento, en sentido amplio, que presentamos consta de tres apartados:

① El desrefinamiento geométrico.

② La renumeración de ecuaciones.

③ La compactación de niveles.

2.6.4. El método multimalla

Se ha dicho ya que una de las ventajas que tiene el utilizar mallados estructurados en el método de los elementos finitos consiste en la facilidad de emplear con ellos métodos multimalla para resolver el sistema de ecuaciones asociado. Como es conocido estos métodos buscan aprovechar el hecho de que en la aplicación de métodos iterativos de resolución de sistemas de ecuaciones la convergencia es rápida con respecto a las altas frecuencias, pero las bajas frecuencias producen una convergencia lenta. Los métodos multimalla eliminan el error de frecuencias bajas en sistemas pequeños, definidos por mallas groseras, resolviéndolos en términos de residuos.

Dado un sistema de ecuaciones:

$$Au = f$$

y una secuencia de mallas encajadas compuesta por ℓ niveles distintos, se pueden resumir los métodos multimalla como sigue:

a) Situados en la malla más fina τ_ℓ, se realiza un presuavizado, es decir se resuelve mediante un método iterativo con un número fijo y reducido de iteraciones, v_1, el sistema de ecuaciones correspondiente a ese nivel de malla:

$$u_\ell := \mathcal{S}_\ell (u_\ell , f_\ell)$$

b) Descendemos al nivel de malla inmediatamente inferior aplicando una restricción, r, al residuo:

$$f_{\ell -1} := r(f_\ell - A_\ell u_\ell)$$

y se aplican γ iteraciones del algoritmo multimalla al sistema:

$$A_{\ell -1} u_{\ell -1} = f_{\ell -1}.$$

En la malla menos fina, es decir la malla inicial, resolvemos de forma exacta:

$$u_0 = A^{-1} f_0$$

c) Una vez obtenido $u_{\ell -1}$, prolongamos a la malla ℓ y corregimos:

$$u_\ell := u_\ell + p\, u_{\ell -1}$$

d) Realizamos v_2 iteraciones de suavizado en la malla ℓ (post-suavizado):

$$u_\ell := \mathcal{S}_\ell\ (u_\ell, f_\ell)$$

En la práctica se utilizan el caso $\gamma = 1$, llamado ciclo V, y el caso $\gamma = 2$, llamado ciclo W; un esquema de este último se muestra en la figura.

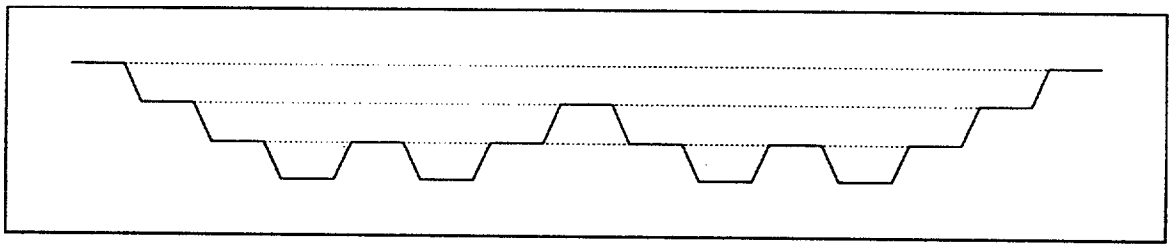


Figura 23.- El ciclo W en una secuencia de 4 niveles de malla.

Podemos suponer que en la resolución de los problemas indicados en los apartados anteriores se dan las siguientes cotas, donde n_ℓ es el número de incógnitas y de ecuaciones del nivel ℓ:

$$u_\ell := \mathcal{S}_\ell\ (u_\ell, f_\ell) \quad \le C_s \cdot n_\ell$$

$$f_{\ell-1} := r(f_\ell - A_\ell u_\ell) \quad \le C_d \cdot n_\ell$$

$$u_\ell := u_\ell + p\, u_{\ell-1} \quad \le C_c \cdot n_\ell$$

$$u_0 = A^{-1} f_0 \quad \le C_0$$

Es decir, las cotas son proporcionales al número de incógnitas puesto que las matrices A_ℓ son uniformemente *sparse*.

Sea $C_H := \sup_{\ell \ge 1} \frac{n_{\ell-1}}{n_\ell}$ y γ tal que $\gamma = 1$ para el ciclo V y $\gamma = 2$ para el ciclo W, entonces, como se sabe, una iteración multimalla en el nivel ℓ necesita un número de operaciones de $C_\ell \cdot n_\ell$, donde:

$$C_\ell \le \frac{\nu C_s + C_d + C_c}{1-\theta} + \theta^\ell \left[\frac{C_0}{\gamma C_H^\ell} + \frac{\nu C_s + C_d + C_c}{1-\theta} \right]$$

es decir

$$C_\ell \le \frac{\nu C_s + C_d + C_c}{1-\theta} + \theta^{\ell-1} \cdot C_o'$$

donde $C'_0 = C_0 / n_\ell$ y $\nu = \nu_1 + \nu_2$ es el número total de iteraciones de suavizado.

Además la cota $C_\ell \cdot n_\ell$ es del orden de $n_\ell \cdot \log_2 n_\ell$.

Otra variante del método multimalla consiste en la técnica de la iteración anidada (*full-multi-grid method*), que produce una aproximación con error del mismo orden que el error de discretización y orden de operaciones $O(n_\ell)$.

Si despreciamos los términos de orden (θ^ℓ) tenemos una estimación del trabajo computacional para una iteración multimalla al nivel K así:

$$W_k = \frac{\nu C_s + C_d + C_c}{1-\theta} n_k$$

Mientras que en el caso de usar la iteración anidada se tiene una expresión:

$$iW_1 + iW_2 + \ldots + iW_\ell \le i \sum_{k=1}^{\ell} C_H^{\ell-k} W_\ell < \frac{i}{1-C_H} W_\ell$$

siendo i el número de iteraciones en cada nivel que suponemos constante. En lo último no se ha tenido en cuenta el trabajo computacional requerido para la resolución del sistema $A_0\, u_0 = f_0$ ni el debido a las restricciones y prolongaciones, todos ellos despreciables con respecto al de los suavizados.

Las dos últimas expresiones son interesantes porque si suponemos que estamos utilizando el ciclo V, $\gamma = 1$, C_H será desde luego menor que la unidad, sin embargo para valores de $n_{\ell-1}/n_\ell$ cercanos a la unidad el trabajo computacional resulta muy grande.

Como se ha comentado anteriormente la reiteración de gran número de refinamientos y desrefinamientos puede hacer aparecer secuencias de mallas en las que cada vez hay más niveles de malla, mientras que el número total de nodos permanece prácticamente constante. Obsérvese que, según lo

anterior, tendríamos un mal comportamiento, en cuanto a tiempo de cálculo, del método multimalla.

Este problema lo hemos resuelto forzando al método a seguir "descendiendo" ó "ascendiendo" en la secuencia de mallas mientras que el cociente entre números de ecuaciones sea superior a un número prefijado. Es decir, si la última malla en la que se realizaron las iteraciones de suavizado fue la de orden K, se seguirá descendiendo en la secuencia de mallas mientras que se tenga $n_{k-r}/n_k < 1/2$, ó bien $n_{k-r}/n_k < 1/3$. Hemos denominado a esta técnica método multimalla *con salto* porque los suavizados no se producen en niveles consecutivos sino que hay, o puede haber, un salto entre dos de tales niveles.

Resultados obtenidos de esta forma se muestran en el capítulo de Aplicaciones.

2.6.5. Comparación con otros algoritmos

En principio, dado un algoritmo de refinamiento en mallados estructurados, se puede pensar y desarrollar el algoritmo de desrefinamiento inverso. Este último podrá ser más o menos complejo. De cualquier forma, el principal, y hasta donde nosotros conocemos único, antecedente del Algoritmo de Desrefinamiento que es la parte fundamental de este trabajo de Tesis es el descrito por M. C. Rivara en [Rivar89]. Sin embargo, se pueden destacar algunas diferencias entre ambos algoritmos.

1) La condición de desrefinamiento que usamos es por nodos, y no por elementos como se sugiere en [Rivar89]. La condición que nos permite decidir cuándo debe eliminarse un nodo es de fácil evaluación y se ha revelado muy eficaz en la práctica; por otra parte, se realiza en un número mínimo de nodos atendiendo al carácter de encajamiento que tienen los mallados con los que trabajamos. En resumen, en el algoritmo aquí desarrollado se eliminan nodos; en el algoritmo descrito en [Rivar89] se eliminan elementos.

2) La estructura de datos aquí utilizada y anteriormente descrita es esencialmente diferente a la estructura molecular utilizada en el trabajo de Rivara. Esta diferencia hace también que el algoritmo aquí presentado sea notablemente distinto en lo que concierne a su puesta en práctica. La estructura de los programas clásicos de elementos finitos se adaptan bien a la estructura de datos aquí empleada, por ello la incorporación del algoritmo de refinamiento y desrefinamiento a un programa estándar es en principio factible.

3) Nuestro algoritmo de desrefinamiento se puede combinar tanto con el algoritmo 2-T como con el 4-T de Rivara. En ambos casos, dependiendo del problema, se pueden conseguir zonas de desrefinamiento muy locales. No tenemos el problema señalado por Rivara de la extensión de la zona desrefinada cuando utiliza el 4-T a la hora de refinar. En este sentido, a diferencia de Rivara, nuestro algoritmo logra la conformidad disminuyendo la zona desrefinada, no extendiéndola.

4) Nuestro algoritmo está completado por el algoritmo de compactación de niveles que hace posible que el número de niveles de malla no crezca excesivamente al tratar problemas evolutivos. Además, la resolución del

problema algebraico asociado al desrefinamiento geométrico, nos permite conservar la mejor aproximación de la solución numérica anterior para esa malla desrefinada y, que la resolución del sistema de ecuaciones sea fácil. Por ese motivo, la incorporación del algoritmo de desrefinamiento ha afectado poco al módulo de resolución del sistema de ecuaciones.

5) Por último, se puede destacar que se ha aplicado el algoritmo de desrefinamiento a diversos problemas como son: problemas estacionarios, quasi-evolutivos y evolutivos.

2.6.6. Complejidad y eficiencia del algoritmo

No es fácil estudiar la complejidad y eficiencia de un determinado algoritmo. (Sobre la complejidad concreta y abstracta de algoritmos puede verse, por ejemplo, [Diaz87]. Unos buenos apuntes sobre el cálculo de la eficiencia de los algoritmos son los de J.L. Balcázar, [Balca92]. Algunos conceptos se encuentran en el apéndice a2).

En nuestro caso hemos estimado el orden de operaciones que realiza el algoritmo de desrefinamiento en función de dos parámetros: el número de nodos en el mallado $NUMP$ y el número de niveles de malla n. La razón de esta hipótesis es que, según se ha observado experimentalmente, dependiendo del problema, el número de niveles de malla pueden crecer cuando se realizan muchos refinamientos/desrefinamientos. De hecho en problemas evolutivos, en los que este número llega a ser de varios miles, el número de niveles de malla ha llegado a superar el centenar.

Para el análisis de algoritmos el caso más usado comúnmente es el análisis del *caso peor*. Este presenta la desventaja de su excesivo pesimismo, dado que probablemente, el comportamiento real del programa será algo mejor que el descrito por el análisis. El caso mejor suele ser irreal (por demasiado optimista) para ser de utilidad práctica. El caso medio corresponde al cálculo de una media del uso de recursos que, como es lógico, debe ser ponderada por la frecuencia de aparición de cada caso; es decir, requiere el cálculo de una esperanza matemática a partir de una distribución de probabilidad. Este análisis proporciona datos mucho más interesantes sobre el algoritmo, pero requiere a su vez más datos (la distribución de probabilidad) sobre el entorno en que el algoritmo se ejecuta, de los que normalmente se carece. Como el algoritmo de desrefinamiento es un algoritmo de propósito general en cuanto a su aplicación a diferentes problemas, hemos estimado el orden de operaciones analizando el diagrama de flujo del mismo y considerando el caso peor. Obtendremos por tanto una estimación de la complejidad del algoritmo como cota superior.

El algoritmo de desrefinamiento consta básicamente de dos bucles; uno, externo, en niveles de malla (desde la malla más fina hasta la segunda), y otro interno, en nodos propios de cada nivel. Prescindiremos del hecho de

que se recorran caras o elementos, pues estas operaciones son del orden del número de nodos.

Teniendo en cuenta el esquema del algoritmo (más detallado se encuentra en el apartado 2.5.) y que el número de nodos propios de la malla τ_j se denotaba por $NNP(j)$, se tiene el siguiente esquema en cuanto a número de operaciones:

Sea la secuencia de entrada: $T = \{ \tau_1 \leq \tau_2 \leq ... \leq \tau_n \}$

Bucle en niveles de malla: $O(n)$

1. Recorrido de los nodos propios de τ_j y evaluación de la condición de desrefinamiento. $O(NNP(j))$

2. Se asegura la conformidad. $O(NUMP \cdot NNP(j))$

   Ahora se definen nuevas conexiones nodales para el naciente nivel de malla, o ese nivel se elimina de los vectores de estructura, en ambos casos el número de operaciones que se realizarán será del orden: $O(NUMP)$

   Por último: los cambios se heredan a los siguientes niveles de malla. $O(n \cdot NUMP)$

   Fin.

La estimación del orden de operaciones del segundo paso merece una explicación detallada. Para entender la fórmula $O(NUMP \cdot NNP(j))$ hay que recordar cómo se aseguraba la conformidad del nivel de malla que se está creando. Véase a este respecto el *procedimiento de Conformidad* en el apartado 2.5.1. Si algún nodo de la malla τ_j permanece, la conformidad se asegura recorriendo los elementos del nivel anterior, es decir de τ_{j-1} y logrando la conformidad local en cada uno de ellos. Si en este proceso se cambia el indicador de desrefinamiento de algún nodo propio de τ_j, hay que realizar un nuevo recorrido de los elementos de τ_{j-1} y asegurar de nuevo la

conformidad. Por esta razón, y teniendo en cuenta que el número de elementos de la malla τ_{j-1} lo podemos suponer del orden de $NUMP$, asegurar la conformidad tendrá un coste no superior a $NUMP \cdot NNP(j)$.

Los pasos 1. y 2. se pueden hacer independientes del bucle en niveles de malla, pues cada nodo sólo se evalúa, como mucho una vez en todo el algoritmo. Es decir, sabiendo que

$$\sum_{j=2}^{n} NNP(j) \leq NUMP$$

se tiene:

$$\mathrm{O}\left(n \cdot NNP(j)\right) = \mathrm{O}(NUMP)$$
$$\mathrm{O}\left(n \cdot NUMP \cdot NNP(j)\right) = \mathrm{O}\left(NUMP^2\right)$$

Por tanto, el algoritmo tiene una complejidad del orden de

$$\mathrm{O}(NUMP^2 + NUMP + n^2 \cdot NUMP) = \mathrm{O}(NUMP^2 + n^2 \cdot NUMP) \quad (1)$$

Si está asegurado que el número de niveles de malla es constante, el análisis anterior nos dice que la complejidad del algoritmo es el orden del número de nodos al cuadrado.

Como se ve, el punto débil del algoritmo es el *procedimiento de conformidad*. Se puede proponer alguna alternativa mediante la que se logre un complejidad de

$$\mathrm{O}(NUMP + n^2 \cdot NUMP). \quad (2)$$

Esto se puede lograr, por ejemplo, del siguiente modo:

La conformidad se puede asegurar *al mismo tiempo* que se recorren los nodos propios de cada nivel de malla y se evalúa la condición de desrefinamiento, siempre que se haya guardado en un vector, llamémosle $IEL(1{:}2,1{:}NUMF)$, para cada cara, los números de los dos elementos que la comparten. Ir almacenando para cada cara esta información no ofrece demasiadas complicaciones en el proceso de refinamiento. Pues bien, con

esta información, si al evaluar la condición de desrefinamiento en un nodo propio N, éste debe permanecer, a través de $IEX(N)$ sabríamos la *cara-entorno* de N; sea ésta NC. Entonces a través de $IEL(1{:}2, NC)$ conoceríamos los números de los elementos de los que se ha de asegurar la conformidad local. Este proceso nos evita los bucles en los elementos de la malla anterior e implicaría, salvo constantes multiplicativas, tantas operaciones como nodos propios de cada nivel. Es decir, los apartados 1. y 2. del algoritmo tendrían una complejidad de $O(NUMP)$, y por tanto, el algoritmo verificaría la fórmula (2). Además, en este caso, si está asegurado que el número de niveles de malla es acotado, se tendría una complejidad del algoritmo de

$$O(NUMP). \qquad (3)$$

Aún en el caso de que se hubiera desarrollado la idea anterior hay que considerar que el hecho clave que hace que el número de operaciones, en la práctica, dependa de una constante multiplicativa de consideración es que se hayan de manejar las colas de los vectores de estructura. La constante multiplicativa podría mejorarse mediante un tratamiento más cuidadoso de las colas de los vectores de estructura.

2.6.7. Los procesos readaptativos

Para terminar este apartado de observaciones vamos a definir formalmente qué entendemos por un proceso readaptativo. Un proceso readaptativo no es más que la combinación de refinamientos y desrefinamientos con vistas a resolver un problema normalmente dependiente del tiempo. La figura resume un proceso de estas características.

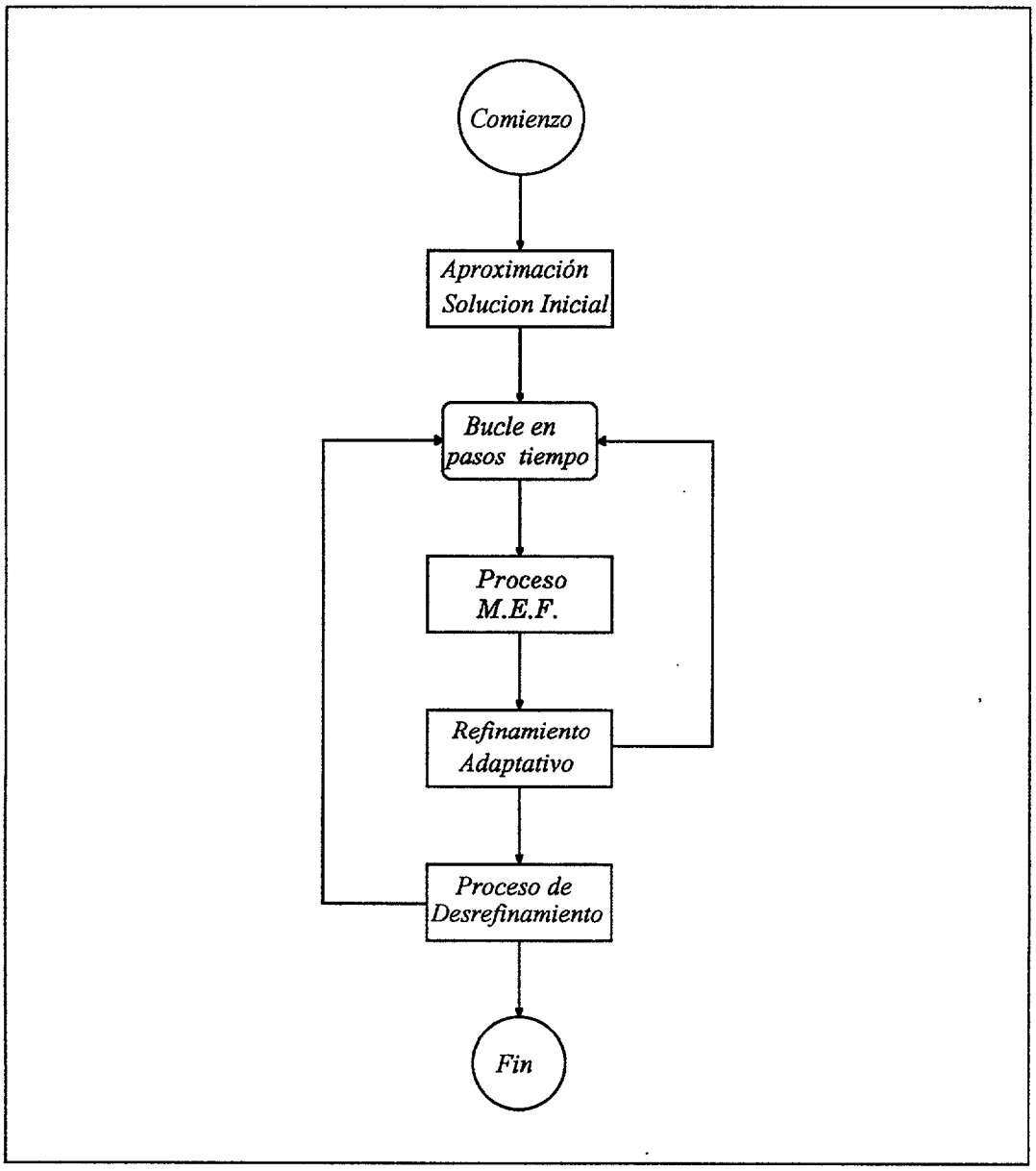


Figura 24.- Módulos principales de un proceso readaptativo.

Con el fin de obtener el mejor soporte para aproximar la solución inicial se realizan algunos refinamientos globales mediante los cuales se localiza suficientemente bien la función. Después se aplica un desrefinamiento con un cierto parámetro ε elegido para optimizar la malla. Este procedimiento para aproximar la solución inicial es muy conveniente para soluciones iniciales con zonas de altos gradientes. De la forma explicada se consigue una malla inicial con un número mínimo de nodos en un tiempo de computación ínfimo. Por tanto sólo es necesario introducir en la entrada de datos una malla capaz de localizar la geometría del dominio.

El número de pasos de tiempo a realizar es fijado por el usuario en la entrada de datos y dependerá de cada problema. En problemas evolutivos puede llegar a ser de varios miles. También se puede utilizar algún criterio de parada cuando el problema ha alcanzado la solución estacionaria.

En cuanto al proceso de Elementos Finitos, éste incluye, si el problema es evolutivo, el cálculo de forma automática del paso de tiempo necesario para asegurar la condición de estabilidad de la formulación para esa malla. En el caso de un problema quasi-evolutivo el paso de tiempo es fijado en la entrada de datos. Finalmente, nuestro código utiliza un método multimalla, dentro del cual se puede elegir de qué forma realizar los suavizados y la resolución del sistema en la malla más grosera. También se fijará un número máximo de iteraciones multimalla para cada resolución. De esta forma se obtiene la solución numérica para la secuencia de mallas en curso.

El usuario fija también la estrategia adaptativa que va ser utilizada, es decir, decide cuántos refinamientos hacer antes de cada desrefinamiento y los parámetros que determinan tanto el refinamiento como el desrefinamiento. Para el refinamiento es necesario determinar: el indicador de error utilizado y el parámetro de refinamiento γ que indica el grado de refinamiento, o los valores mínimo y máximo dependiendo de un número de nodos considerado óptimo en la malla, como se explicó en el apartado 2.1. El desrefinamiento por su parte queda determinado por el parámetro ε, el número de mallas mínimo que no se desrefinará y el indicador de desrefinamiento usado (diferencia relativa o diferencia absoluta). Como se ve, fijados muy pocos parámetros, el código es capaz de adaptar la malla en cada paso de tiempo.

Esto supone una adaptación automática de la malla en cada instante de tiempo, manteniéndose el número de nodos acotado en todo el proceso. En el capítulo siguiente se muestran algunas aplicaciones.

3

Aplicaciones numéricas

En este capítulo se presentan diversas aplicaciones numéricas del algoritmo de desrefinamiento que se ha desarrollado. En primer lugar se tratan algunos problemas estacionarios, en concreto un problema de dos grados de libertad por nodo y después, como muestra de la bondad de este algoritmo para aproximar una función de dos variables con un mallado triangular "mínimo", se muestran los resultados de "desrefinar" una imagen clásica. El segundo apartado muestra varios problemas quasi-evolutivos, entre los que se encuentra el problema modelo inicial. Para finalizar se tratan problemas de convección-difusión con término de convección dominante.

Oportunamente se resaltan algunas características de los métodos readaptativos.

3.1. Problemas estacionarios

Como es conocido, en principio, los problemas estacionarios no son los más apropiados para ser tratados mediante el desrefinamiento. Sin embargo, antes que nada, se puede hacer notar que en aquellos problemas estacionarios en los que no se conozca un indicador (estimador) fiable del error en orden a la implementación del refinamiento local, siempre tendremos la posibilidad de refinar globalmente y después desrefinar de acuerdo con el parámetro ε que elijamos. En este sentido se puede decir que desrefinar después de un refinamiento global equivale a un refinamiento local.

En este apartado, se aplica la técnica anterior en el primer problema. Además esta técnica se ha empleado por el autor para aproximar de forma óptima (con una malla mínima) la solución inicial de problemas evolutivos. También se puede utilizar para localizar las condiciones iniciales o de contorno de cualquier problema con una malla mínima. De esta manera, para determinar la malla inicial en cualquier tipo de problema sólo habría que determinar una malla grosera, pero que fuera capaz de representar el borde del dominio. Para eso se podrían emplear malladores del tipo no estructurado. Sobre esa malla grosera inicial, cuya generación es bastante automática con muy pocos datos de entrada [Cuest92], se puede aplicar la combinación precisa de refinamientos-desrefinamientos para conseguir, también de forma automática, una malla más adaptada a la solución inicial del problema o que represente otro tipo de condiciones sobre la frontera.

3.1.1. Un problema de dos grados de libertad por nodo

En este ejemplo se aplica el algoritmo de desrefinamiento en un problema singular de elasticidad plana estudiado en [MoMoW90]. El dominio y las condiciones de contorno se pueden apreciar en la fig. 25. El problema está definido en una placa rectangular de 500x600 cm^2 con una fractura de 100 cm situada transversalmente en la mitad de uno de los lados mayores. La placa está sometida a tracción por una fuerza de 1 kg/m uniformemente distribuida en sus lados menores. El módulo de Young es $E = 2.10^4$ kg/m^2 y el coeficiente de Poisson $\nu = 0.3$; se supone también que las fuerzas de volumen son nulas.

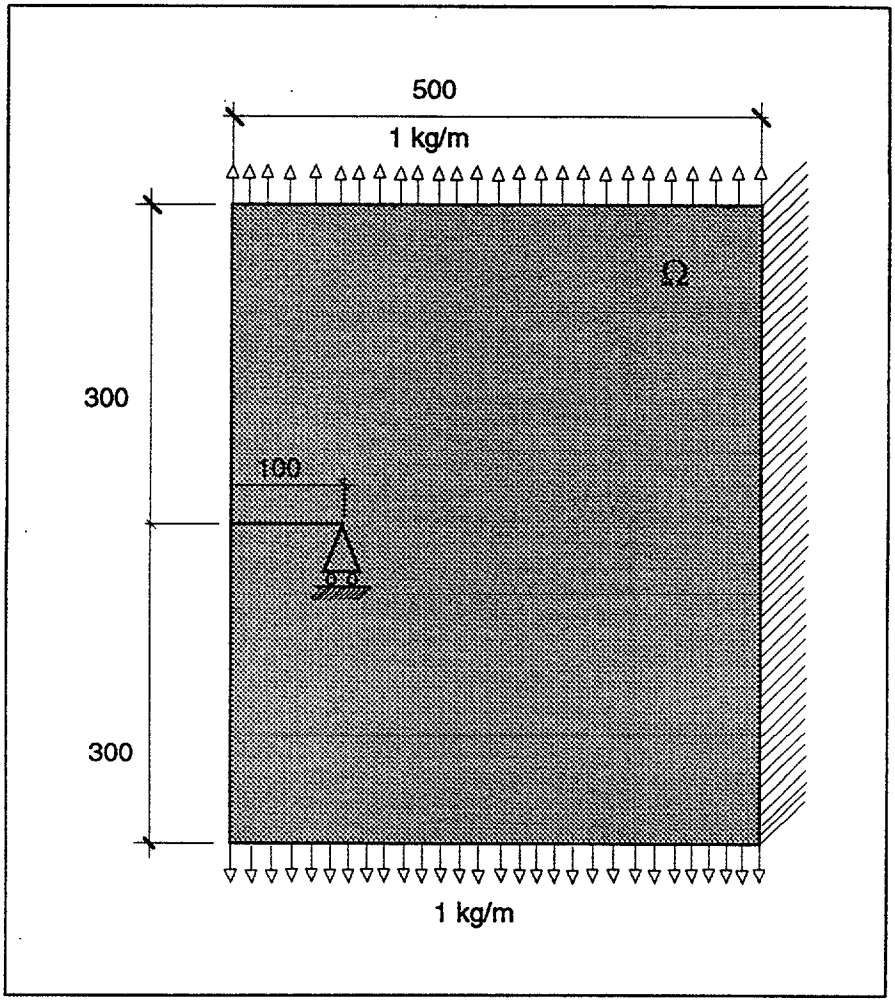


Figura 25.- Dominio y condiciones de contorno.

La formulación del problema es la siguiente:

$$\sum_{j=1}^{2}\frac{\partial}{\partial x_j}\sigma_{ij}(\vec{u})=0 \quad \text{en} \quad \Omega, \quad \text{para i = 1, 2}$$

$$\vec{u}_i=\vec{0} \quad \text{en} \quad \Gamma_0, \qquad\qquad u_2(A)=0$$

$$\sum_{j=1}^{2}\sigma_{1j}(\vec{u})n_j=0 \quad \text{sobre} \quad \Gamma_{1,1}, \qquad \sum_{j=1}^{2}\sigma_{2j}(\vec{u})n_j=1 \quad \text{sobre} \quad \Gamma_{1,1}$$

$$\sum_{j=1}^{2}\sigma_{1j}(\vec{u})n_j=0 \quad \text{sobre} \quad \Gamma_{1,2}, \qquad \sum_{j=1}^{2}\sigma_{2j}(\vec{u})n_j=-1 \quad \text{sobre} \quad \Gamma_{1,2}$$

$$\sum_{j=1}^{2}\sigma_{ij}(\vec{u})n_j=0 \quad \text{para} \quad i=1,2 \quad \text{sobre} \quad \Gamma_{1,3}$$

donde $\vec{u}=(u_1,u_2)$ representa la incógnita, es decir, los desplazamientos, y $\Gamma=\Gamma_0\cup\Gamma_{1,1}\cup\Gamma_{1,2}\cup\Gamma_{1,3}$ es la frontera de Ω. Véase la fig. 26. En la figura puede apreciarse además el mallado inicial que se ha utilizado formado por 5 elementos y 7 nodos.

La estrategia readaptativa seguida para resolver este problema fue: después de cada refinamiento global se aplicó un desrefinamiento con parámetro $\varepsilon=1.5\cdot10^{-6}$ y diferencia absoluta, por cada grado de libertad, como condición de desrefinamiento. Recuérdese que el criterio de desrefinamiento es restrictivo: basta que un grado de libertad no cumpla la condición para que ese nodo deba permanecer.

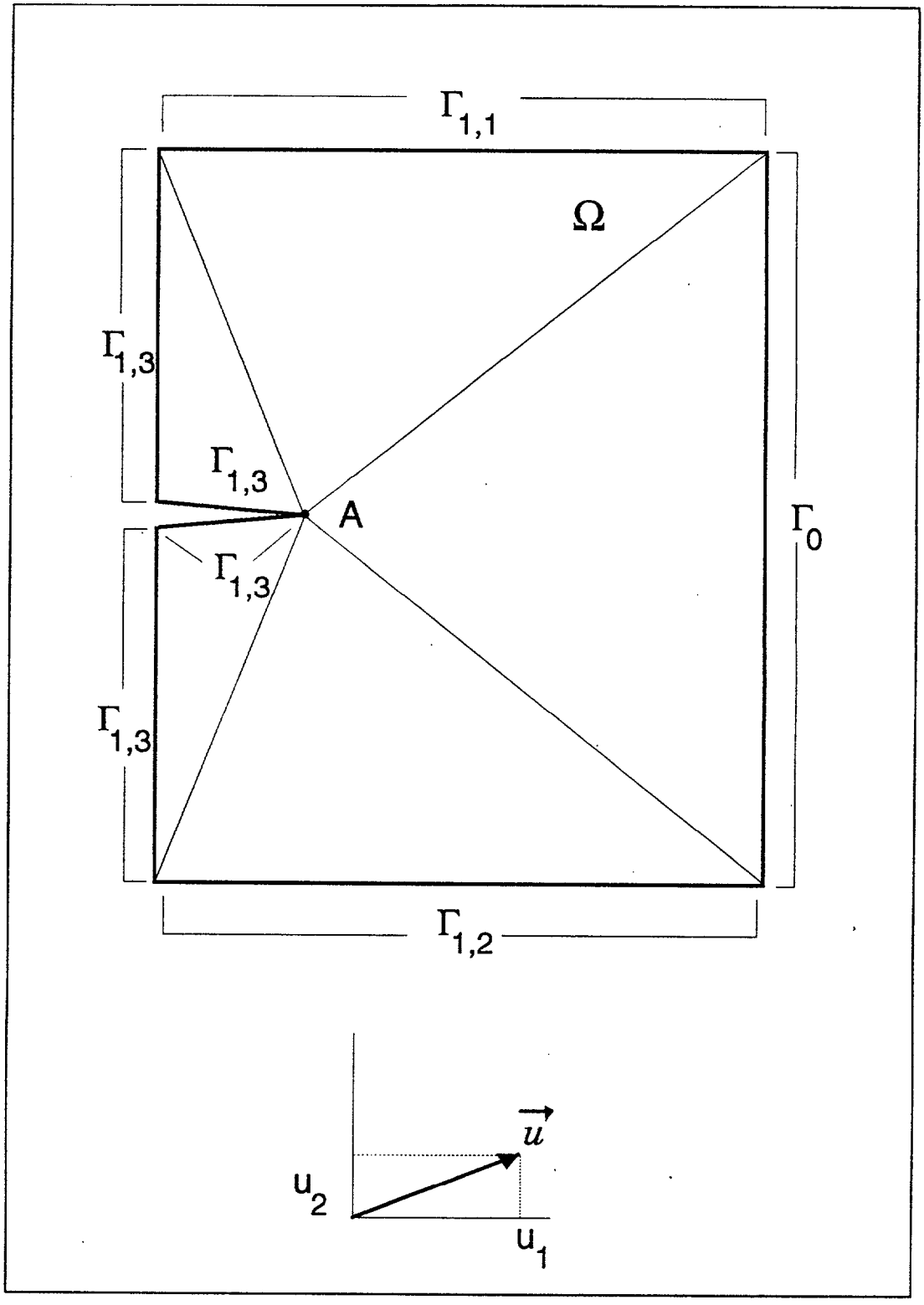
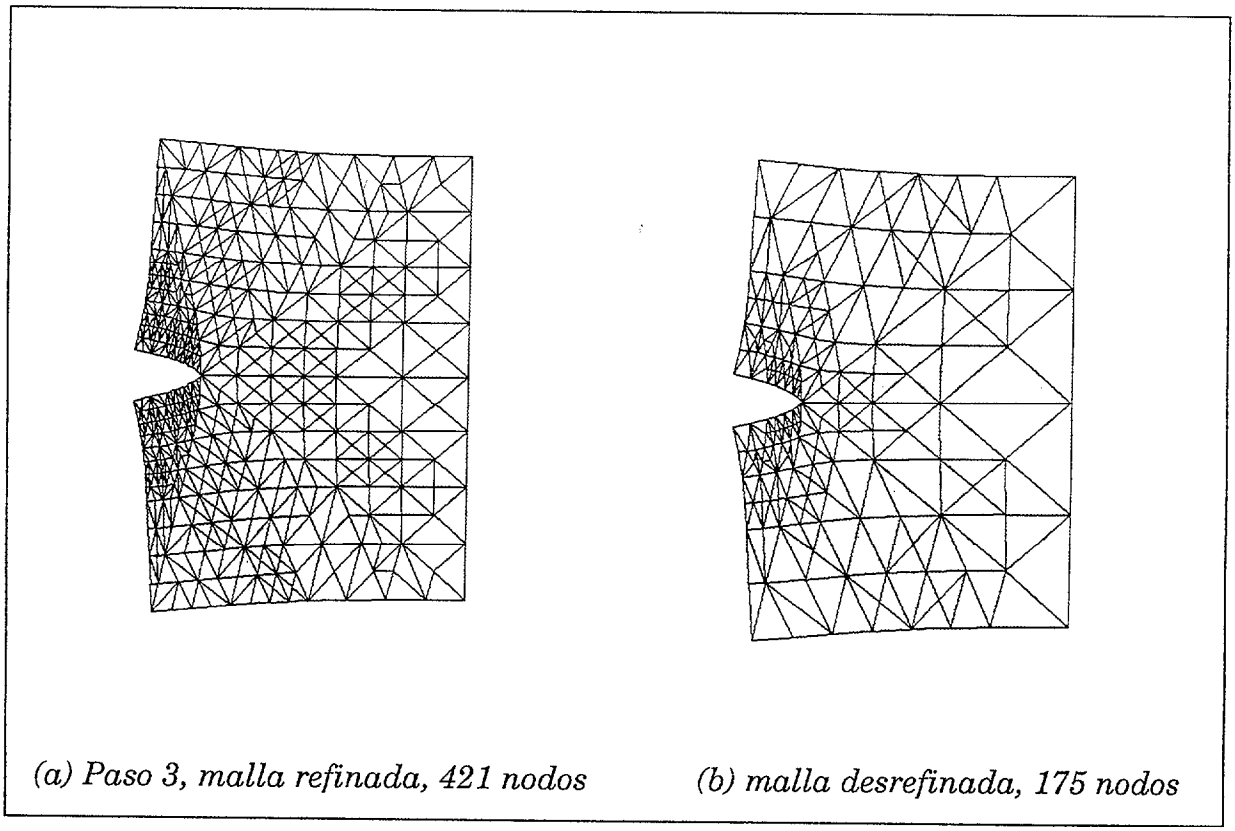


Figura 26.- Mallado inicial.

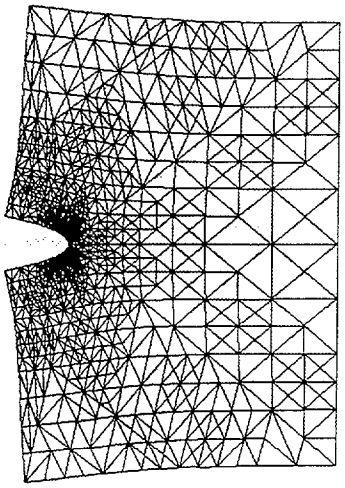
En las páginas siguientes (fig. 27) se muestran algunas mallas refinadas y las correspondientes mallas desrefinadas y la deformación obtenida en la placa a una escala ampliada.

Es importante resaltar cómo el número de nodos se estabiliza tanto en la secuencia de mallas refinadas, como en las desrefinadas. Esto parece sugerir que la solución ya se ha alcanzado con la precisión requerida por el parámetro de desrefinamiento ε. El criterio anterior puede utilizarse para abortar la ejecución de un problema estacionario: si el añadir nodos en el dominio no hace que la nueva solución difiera en algún punto más que el parámetro de desrefinamiento, la ejecución del problema debe parar puesto que se ha alcanzado la solución con la precisión requerida. La evolución de los números de nodos en las distintas mallas se puede observar en la fig. 28.

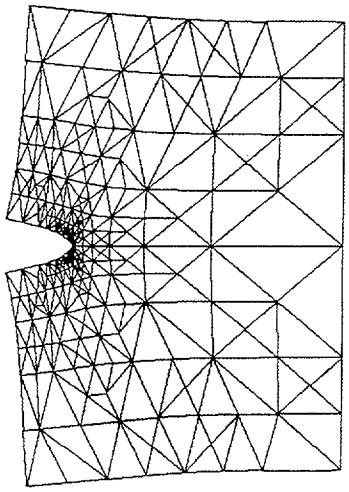
En este problema, la aplicación del método multimalla con gradiente conjugado como suavizado alcanza la solución con un valor estabilizado de doce iteraciones.



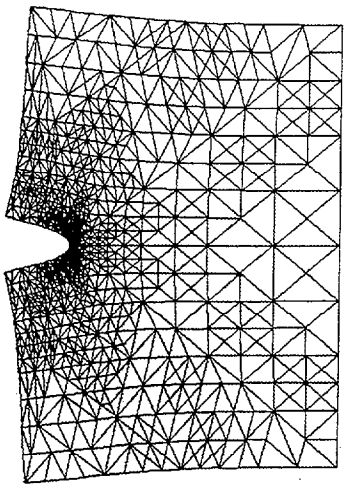
(a) Paso 3, malla refinada, 421 nodos *(b) malla desrefinada, 175 nodos*



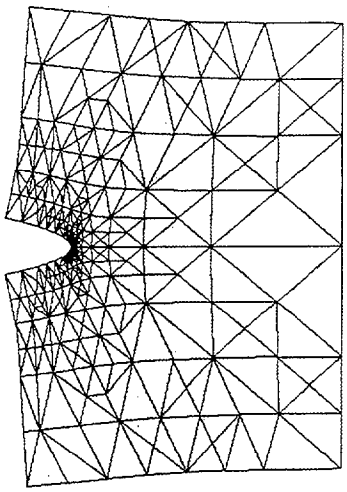
(a) Paso 5, malla refinada, 797 nodos



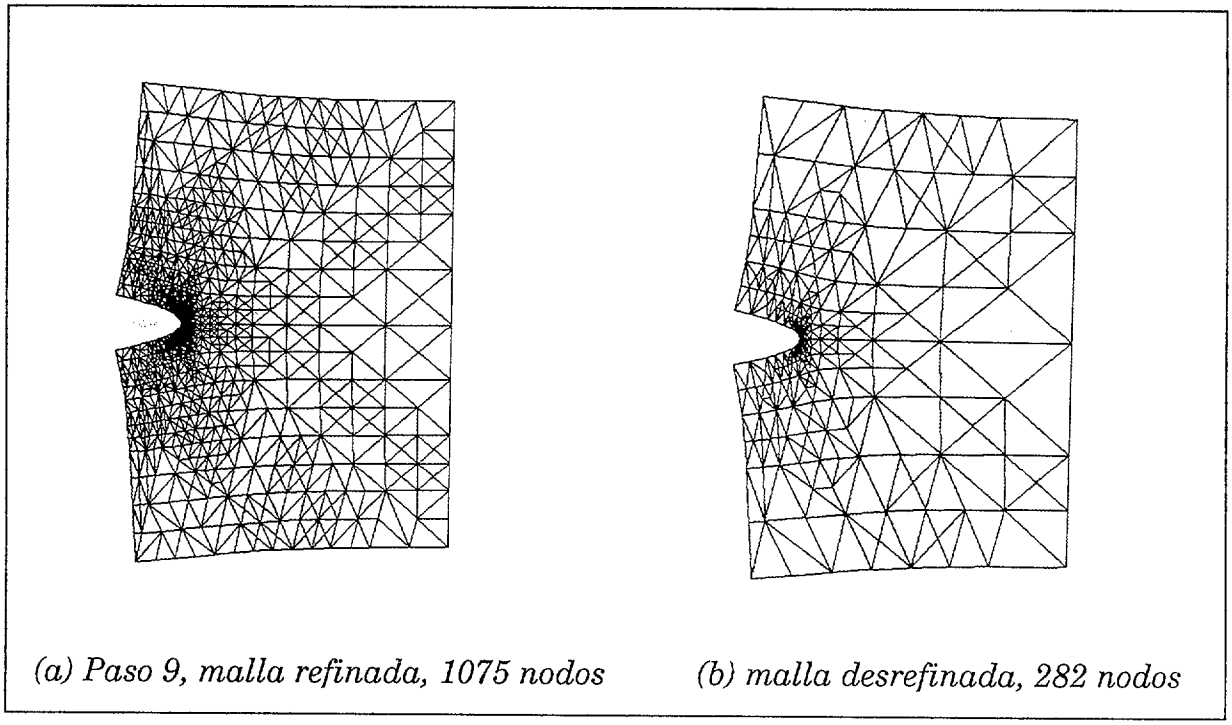
(b) malla desrefinada, 246 nodos



(a) Paso 7, malla refinada, 1001 nodos



(b) malla desrefinada, 273 nodos



(a) Paso 9, malla refinada, 1075 nodos *(b) malla desrefinada, 282 nodos*

Figura 27.- Mallas refinadas y mallas desrefinadas en un problema estacionario, con dos grados de libertad por nodo.

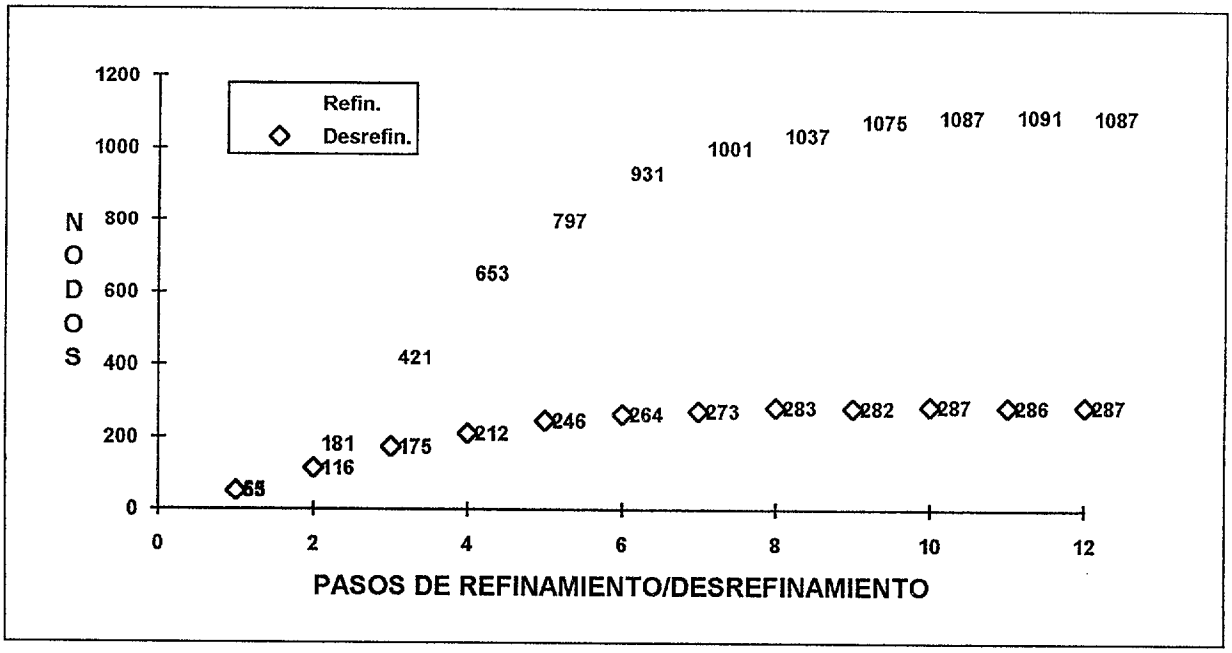


Figura 28.- Mallas refinadas y mallas desrefinadas en un problema estacionario, con dos grados de libertad por nodo.

3.1.2. Aproximación de una función de dos variables

Como segundo ejemplo de aplicación numérica del algoritmo de desrefinamiento a problemas estacionarios, y dentro del marco de la aproximación de funciones de dos variables se presentan los resultados de aplicar diversos desrefinamientos – variando el parámetro ε – a una imagen clásica de 256x256 pixels, en blanco y negro.

Como es conocido, una imagen bidimensional se representa formalmente como una aplicación $I_0(x,y)$ de R^2 en R. Esta aplicación determina el nivel de gris de los pixels si es una imagen en blanco y negro o cualquier otra propiedad. Antes de poder aplicar a una imagen de este estilo nuestro algoritmo de desrefinamiento, hemos de emplear una estructura de mallas encajadas en la que almacenar esa función bidimensional.

Supongamos que tenemos una imagen de 2x2 pixels. Es decir una función de dos variables reales con 4 valores reales: los valores son constantes por pixel. La figura 29 muestra los cuatro pixels originales y una malla triangular asociada. Como nuestro algoritmo trabaja con valores de una cierta función en los vértices de los triángulos, asignamos a los nodos señalados en blanco en la figura el valor de gris del pixel correspondiente.

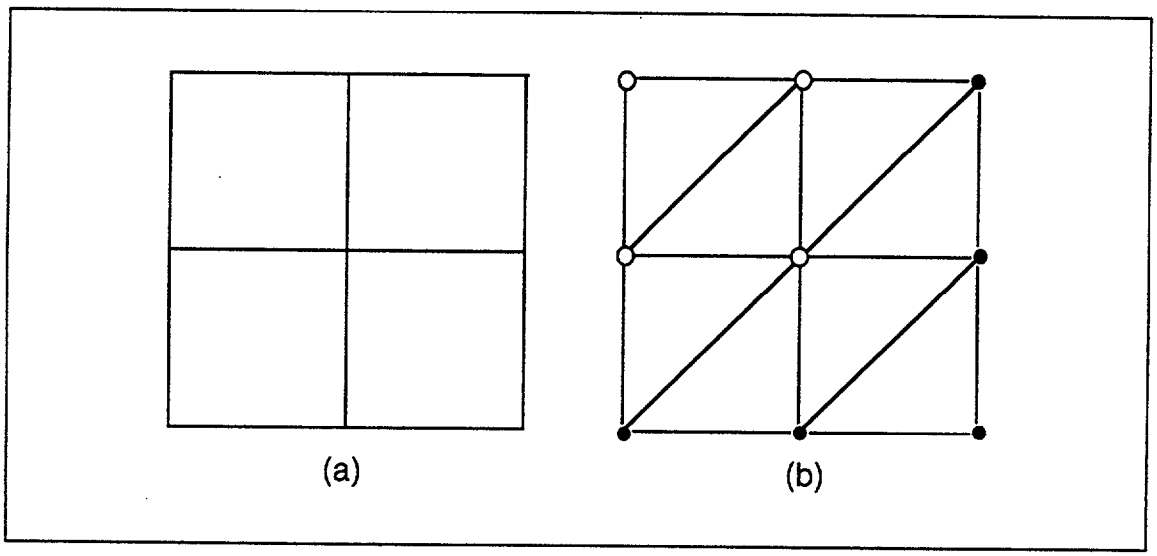


Figura 29.- (a) Imagen de 4 pixels, (b) malla asociada.

Aparecen en la malla triangular de la figura anterior cinco nodos en los que guardamos el mismo valor que en el nodo de la izquierda, para los dos de la derecha, o el de arriba, para los de la fila inferior (nodos en negro en la figura). La utilidad de esa malla estriba en que si se divide cada pixel de la figura anterior en cuatro por segmentos paralelos a los lados, y se repite el proceso, aparece siempre una imagen de $2 \cdot 2^n \times 2 \cdot 2^n$ pixels, donde n es el número de veces que se ha repetido este proceso.

La malla asociada a la primera división de la imagen anterior es la malla resultado de refinar globalmente la malla de la figura. En esa malla también tendríamos la fila inferior y columna derecha de nodos no significativos. Véase la fig. 30. Utilizamos, así la estructura de datos y el algoritmo de refinamiento global del código Neptuno para capturar la imagen original. En la malla de la figura se pueden distinguir los dos niveles de malla, pues las caras del primer nivel están señaladas en trazo grueso.

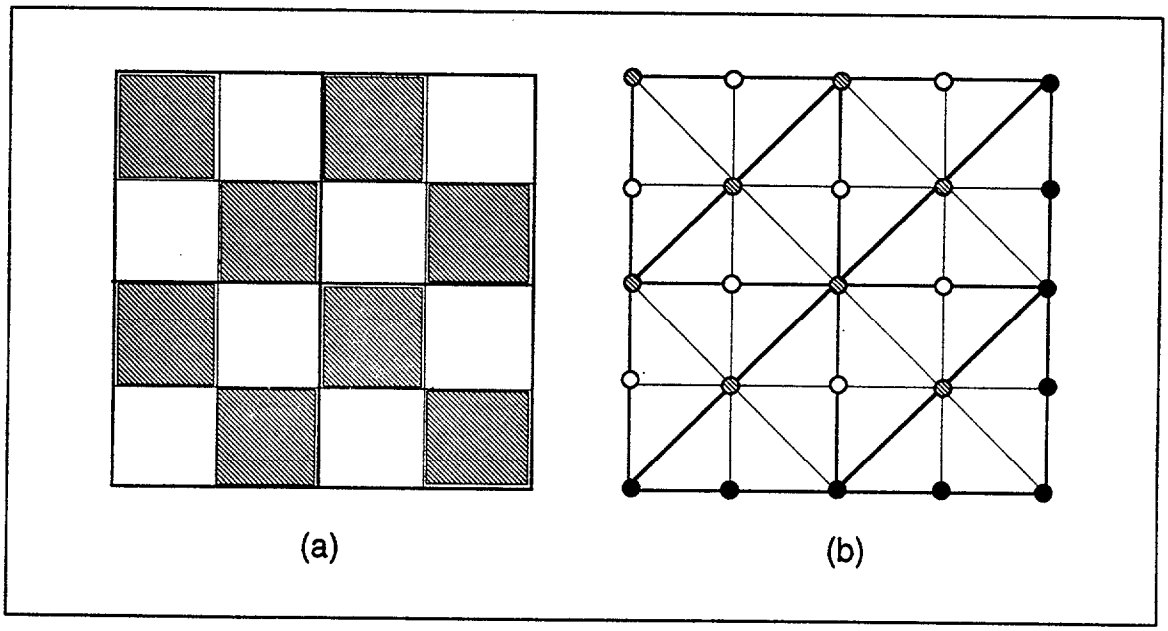


Figura 30.- (a) Imagen de 8 pixels, (b) malla asociada.

Por último, como dentro del amplio campo del Análisis y Proceso de imágenes digitales son muy comunes las imágenes de 256x256, 512x512 ó 1024x1024 pixels, el método descrito anteriormente es apropiado para adjudicar a una imagen una estructura de mallas triangulares anidadas partiendo siempre de la malla grosera de la fig. 29 de forma que sólo hay que determinar cuántos refinamientos globales hay que hacer.

La fig. 31 muestra una conocida imagen 256x256 (véase, por ejemplo, [GerYa85]) a la que, tras formar la secuencia de 8 mallas encajadas, le hemos aplicado diversos desrefinamientos.

Figura 31- Imagen clásica 256x256.

En las páginas siguientes aparecen las mallas triangulares desrefinadas y las imágenes asociadas a cada malla. Estas imágenes no son la imagen original sino las imágenes elaboradas partiendo de la información que contiene la respectiva malla desrefinada y teniendo en cuenta que tenemos la solución interpolada en cada triángulo. Puede observarse cómo hasta un valor de $\varepsilon = 20$ ó $\varepsilon = 30$ es inapreciable, a simple vista, la diferencia entre la imagen original y la imagen "desrefinada".

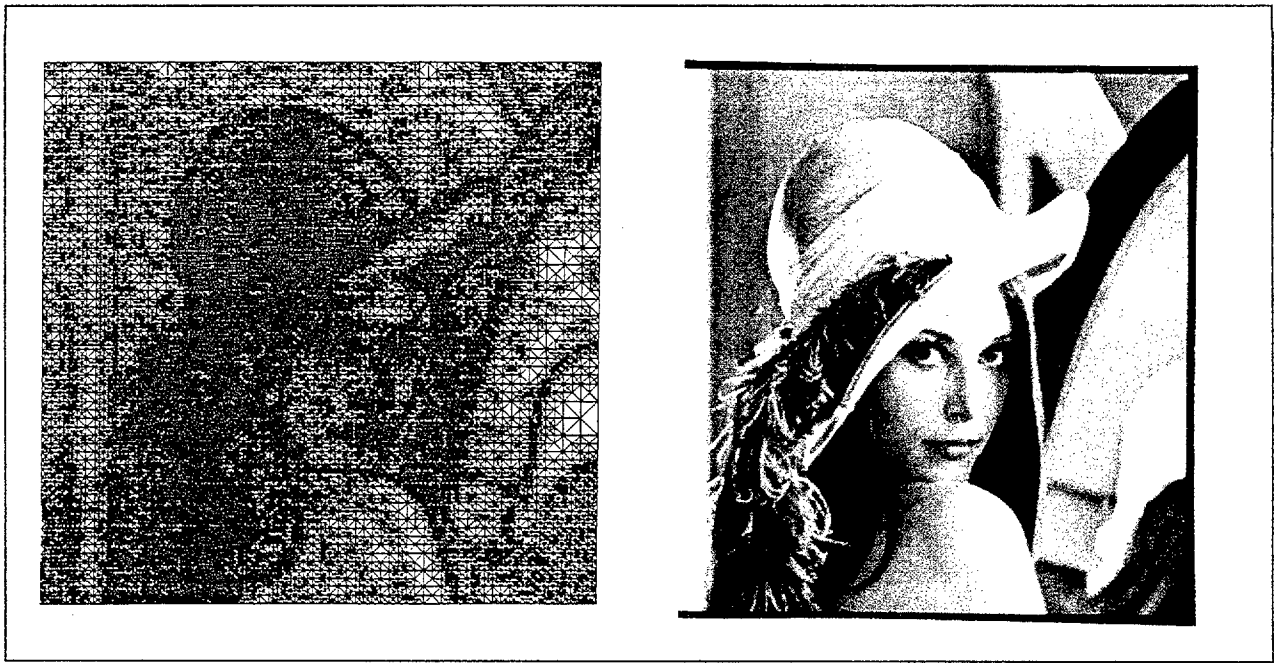


Figura 32.- Malla e imagen para ε = 5: 30894 nodos, SNR(dB) = 36.12.

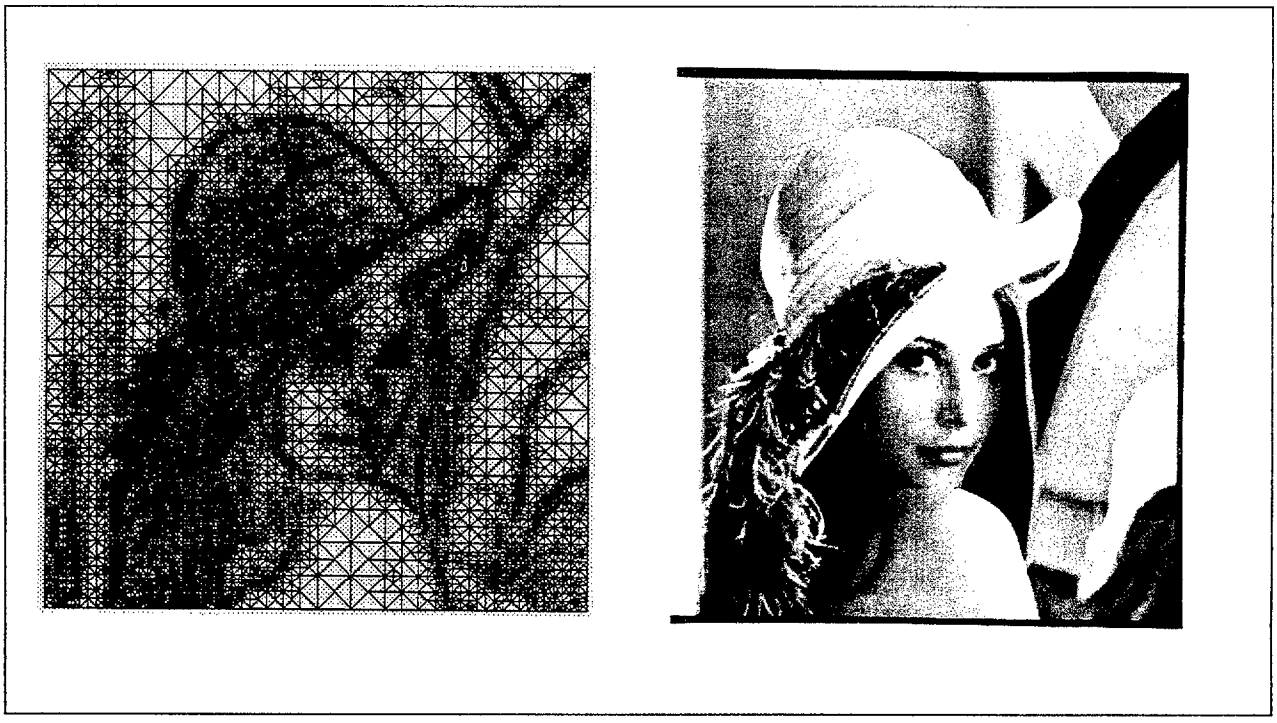


Figura 33.- Malla e imagen para ε = 10: 18.805 nodos, SNR(dB) = 30.67.

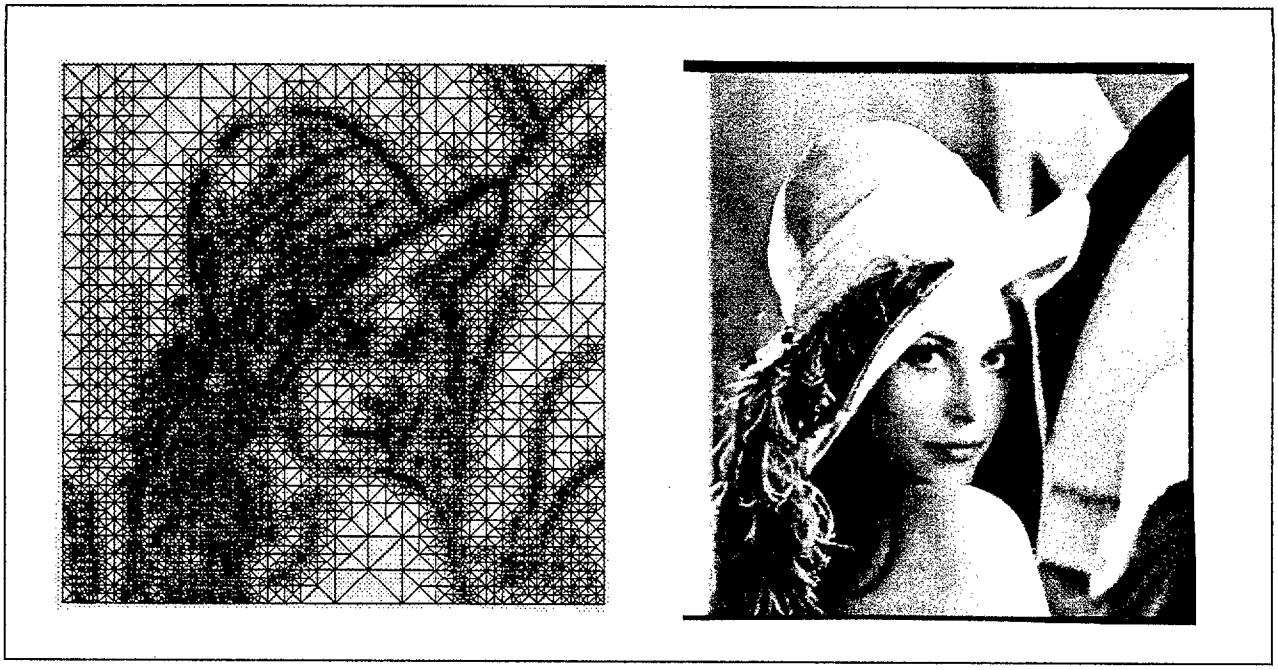


Figura 34.- Malla e imagen para $\varepsilon = 15$: 13.892 nodos, SNR(dB) = 27.53.

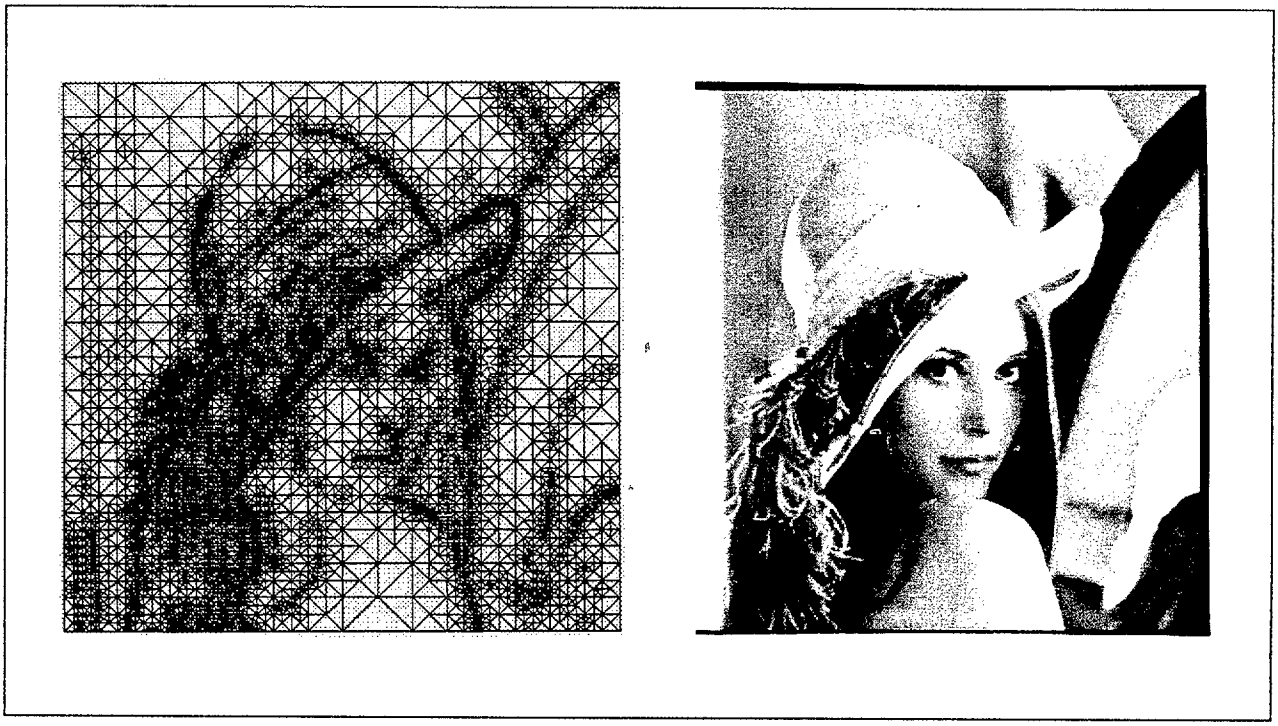


Figura 35.- Malla e imagen para $\varepsilon = 20$: 11.018 nodos, SNR(dB) = 25.45.

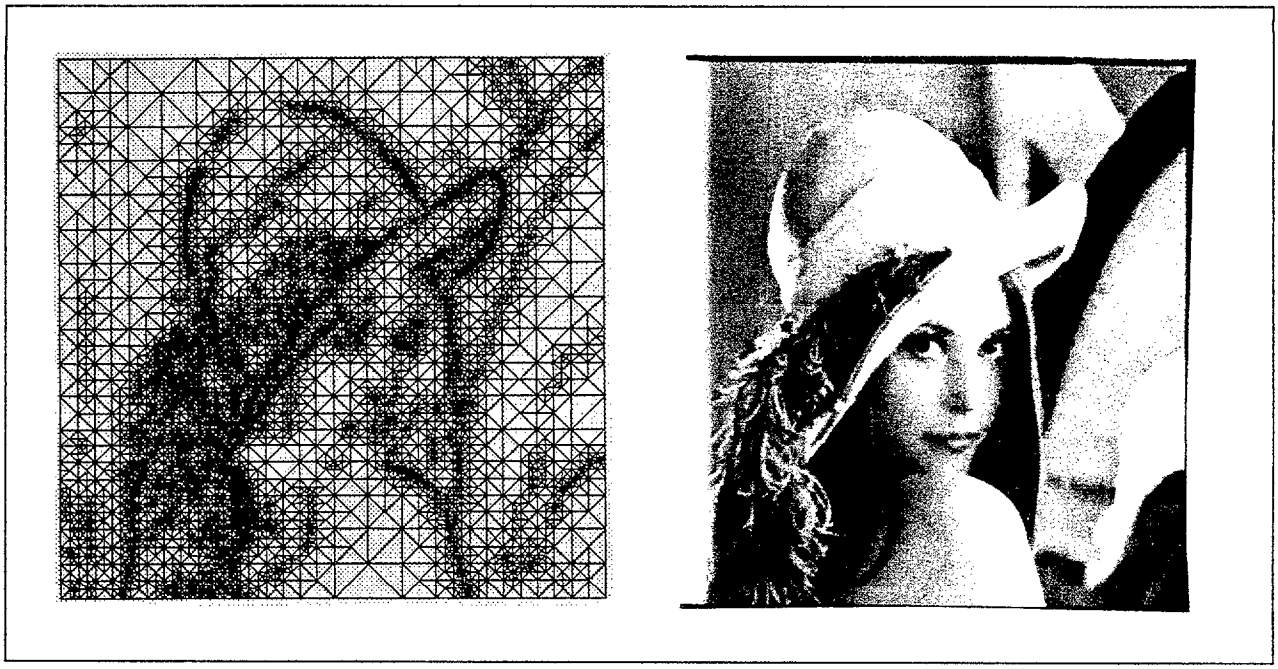


Figura 36.- Malla e imagen para $\varepsilon = 30$: 7.736 nodos, SNR(dB) = 22.58.

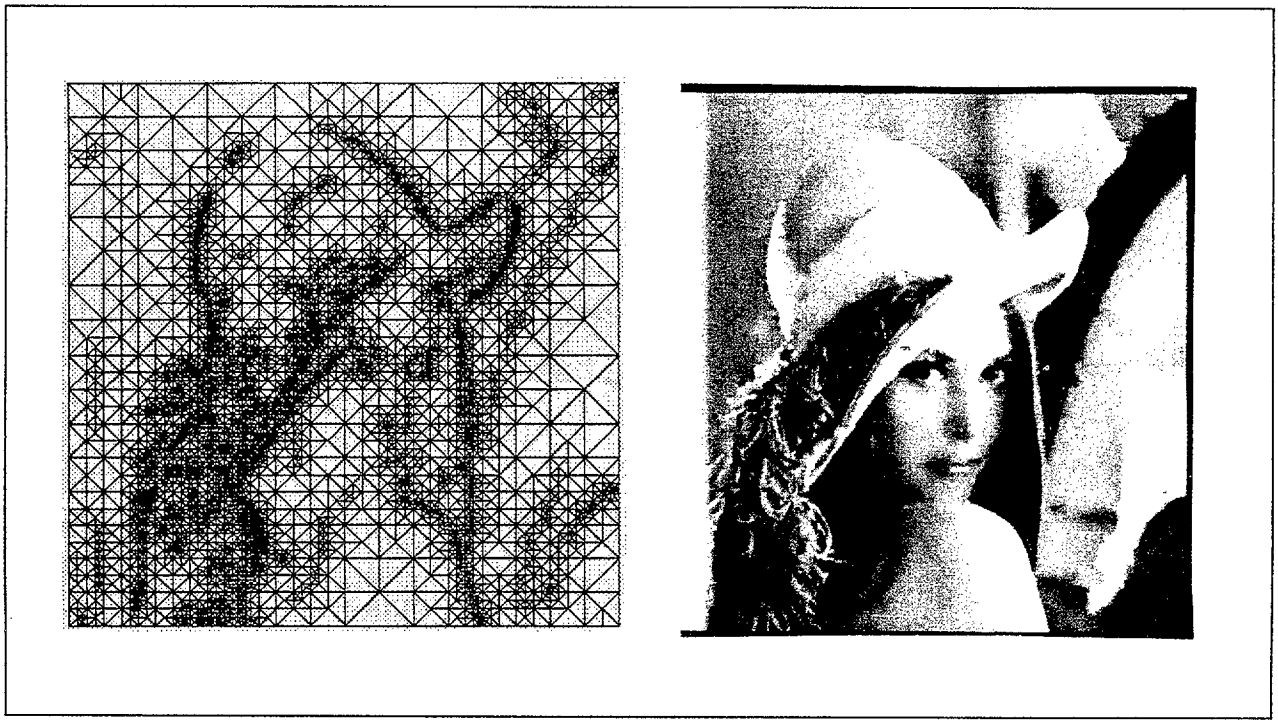


Figura 37.- Malla e imagen para $\varepsilon = 40$: 5.555 nodos, SNR(dB) = 19.95.

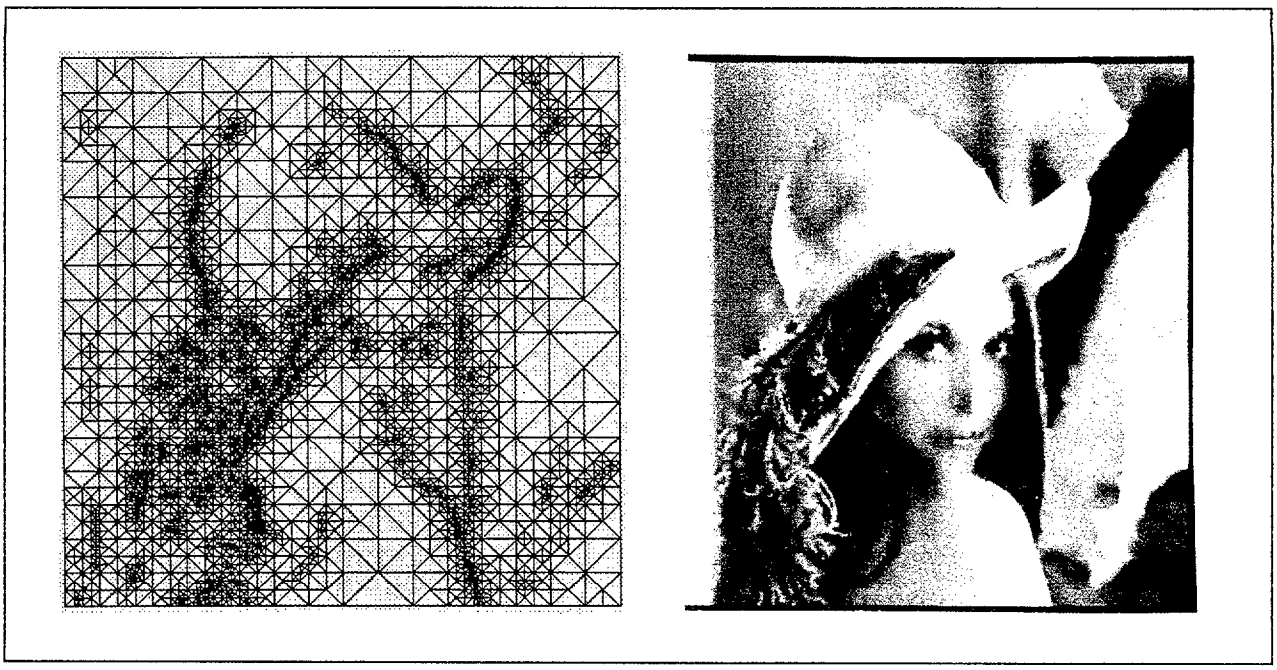


Figura 38.- Malla e imagen para $\varepsilon = 50$: 3.980 nodos, SNR(dB) = 18.17.

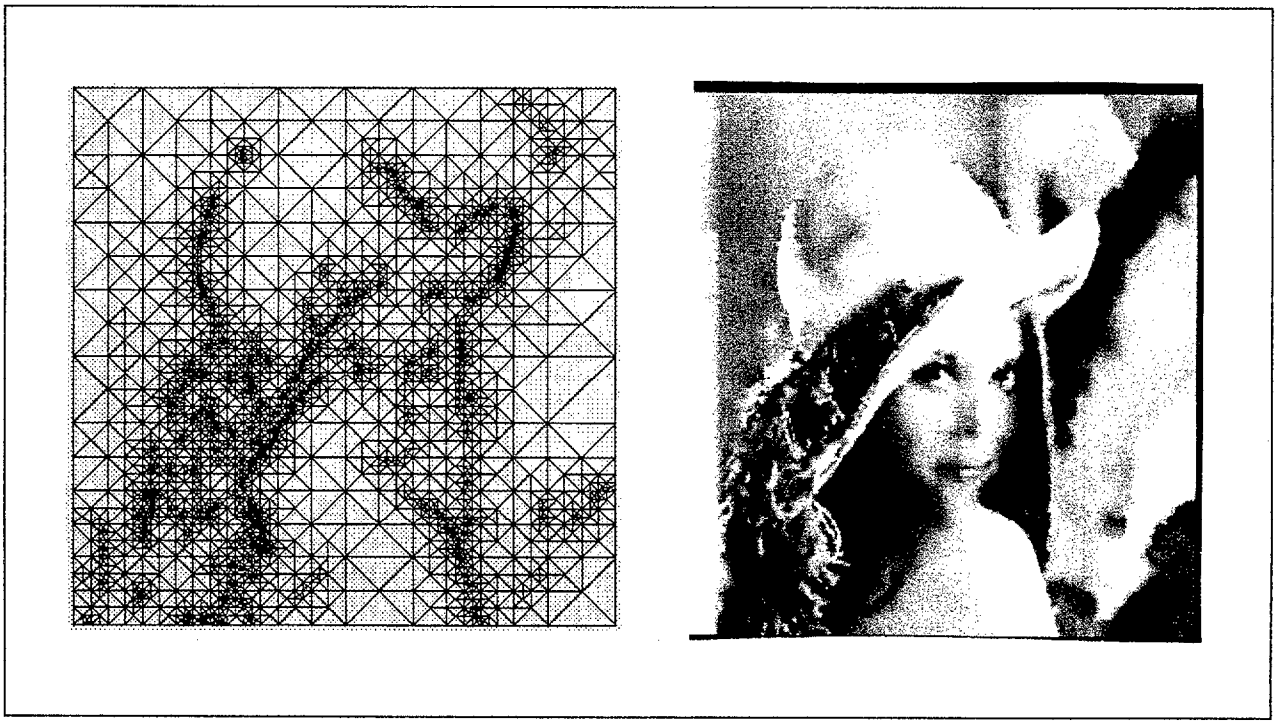


Figura 39.- Malla e imagen para $\varepsilon = 60$: 2.914 nodos, SNR(dB) = 16.06.

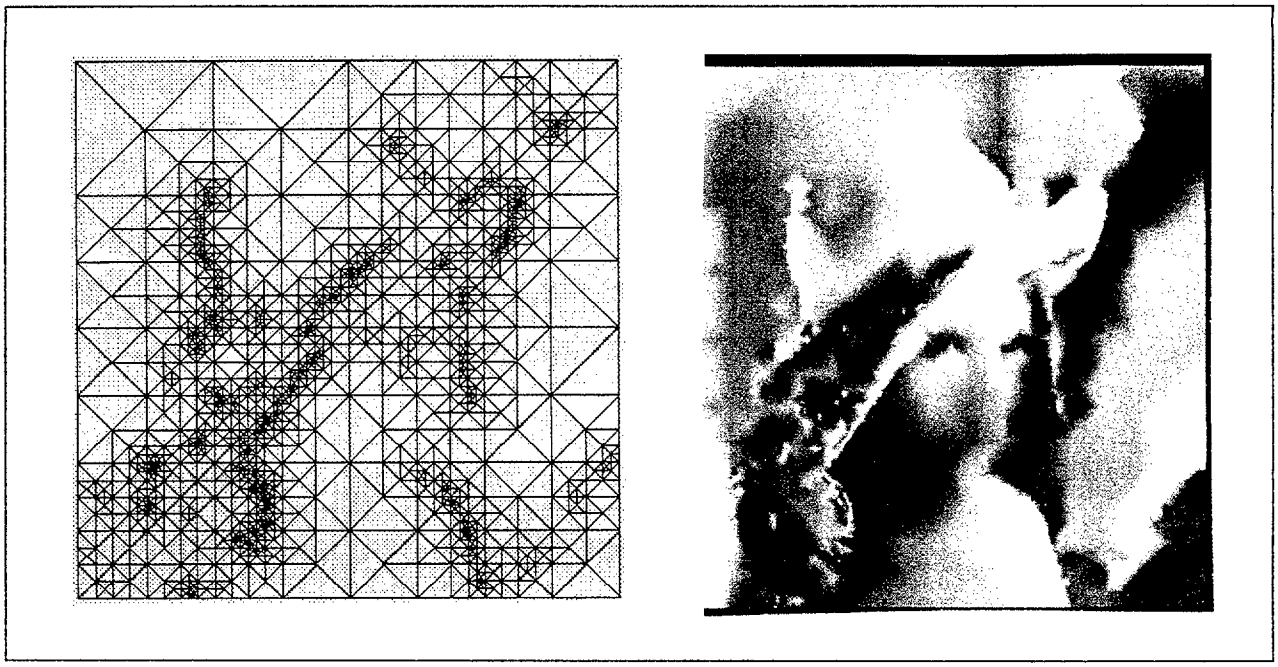


Figura 40.- Malla e imagen para ε = 80, SNR(dB) = 12.89.

Para medir la distorsión ó ruido introducido en la imagen al ser *desrefinada,* se ha utilizado la conocida fórmula de *SNR(db) - signal noise relation, en decibelios-* :

$$SNR = 10 \cdot \log \frac{\sum_{n=1}^{N} s^2(n)}{\sum_{n=1}^{N} e^2(n)}$$

donde N es el número de píxels en la imagen, $s(n)$ es el valor exacto de gris en el píxel n, y $e(n)$ es el error cometido, es decir $e(n) = s(n) - s_i(n)$, la diferencia entre el valor exacto y el aproximado (en este caso interpolado).

En realidad, para una aplicación exhaustiva del algoritmo de desrefinamiento al tratamiento de imágenes habría que replantear el código Neptuno. La estructura de datos no parece la más adecuada para tratar problemas referidos siempre a dominios cuadrados, en los que, por otra parte, con un buen almacenamiento de los datos podrían obviarse, por ejemplo, las coordenadas de los nodos.

Como ya se ha dicho, no se pretende con esta aplicación demostrar la eficacia del algoritmo respecto a los problemas de las imágenes bidimensionales sino mostrar su bondad en relación a la aproximación de funciones de dos variables. En este sentido, parece lógico, si se van a resolver los problemas asociados a una imagen que involucran ecuaciones diferenciales por el método de los elementos finitos aplicar a la imagen original un desrefinamiento previo con ε=15 ó ε=20, a fin de disminuir el número de ecuaciones que habremos de resolver, perdiendo la información menos significativa.

Por poner un ejemplo, si nos referimos al problema del filtrado de imágenes, con objeto de eliminar el ruido, se puede decir lo siguiente:

Dada una imagen $I_0(x,y)$ el método clásico de realizar un filtrado es la convolución de ella con un núcleo gaussiano [Marr82]. A partir de la imagen original se obtiene una familia de imágenes "regularizadas" $I(x,y,\alpha)$ mediante la convolución:

$$I(x,y,\alpha) = \int_{R^2} I_0(x,y) \cdot G(x-s, y-z, \alpha)\, dsdz$$

donde $G(x,y,\alpha)$ es un núcleo gaussiano que tiene la expresión

$$G(x,y,\alpha) = \frac{1}{2\alpha\pi} e^{-\frac{x^2+y^2}{2\alpha}}$$

y donde α es un parámetro a elegir que representa la varianza de la gaussiana y que puede interpretarse como un parámetro de escala.

Este proceso se puede entender en términos de ecuaciones diferenciales en derivadas parciales en el sentido de que la familia de imágenes regularizadas $I(x,y,\alpha)$ representa (salvo una constante) la solución de la ecuación en derivadas parciales de difusión lineal del calor, donde el parámetro α representa el tiempo (α = t), y la función $I_0(x,y)$ la solución inicial. Matemáticamente esto quiere decir que $I(x,y,t)$ es la solución de la ecuación diferencial:

$$\begin{cases} I_t = div(k\nabla I) & en \quad R^2 \times R^+ \\ I(x,y,0) = I_0(x,y) & en \quad R^2 \end{cases}$$

donde I_t representa la derivada de I respecto al tiempo, $div(\cdot)$ el operador divergencia, $\nabla(\cdot)$ el operador gradiente y k la conductividad del medio, que en este caso es una constante positiva. El problema no está completo si no se especifican las condiciones de contorno. Estas suelen ser de flujo nulo en la frontera.

Con el fin de disminuir el número de incógnitas que aparecen en el problema es muy conveniente aplicar, antes de utilizar un proceso de elementos finitos para resolverlo, un desrefinamiento a la solución inicial, es decir, a la imagen original, de forma semejante a como hemos hecho en los problemas evolutivos que se muestran en el tercer apartado de este capítulo.

3.2. Problemas quasi-evolutivos

Presentamos aquí algunos ejemplos de aplicación de un proceso readaptativo a la resolución de ecuaciones diferenciales en las que la solución no depende "intrínsecamente" del tiempo, es decir, no aparece ninguna derivada parcial de la solución respecto del tiempo, y sin embargo el tiempo aparece como un parámetro en la función del segundo miembro. Nos interesa conocer la familia de soluciones correspondiente a esta familia de ecuaciones.

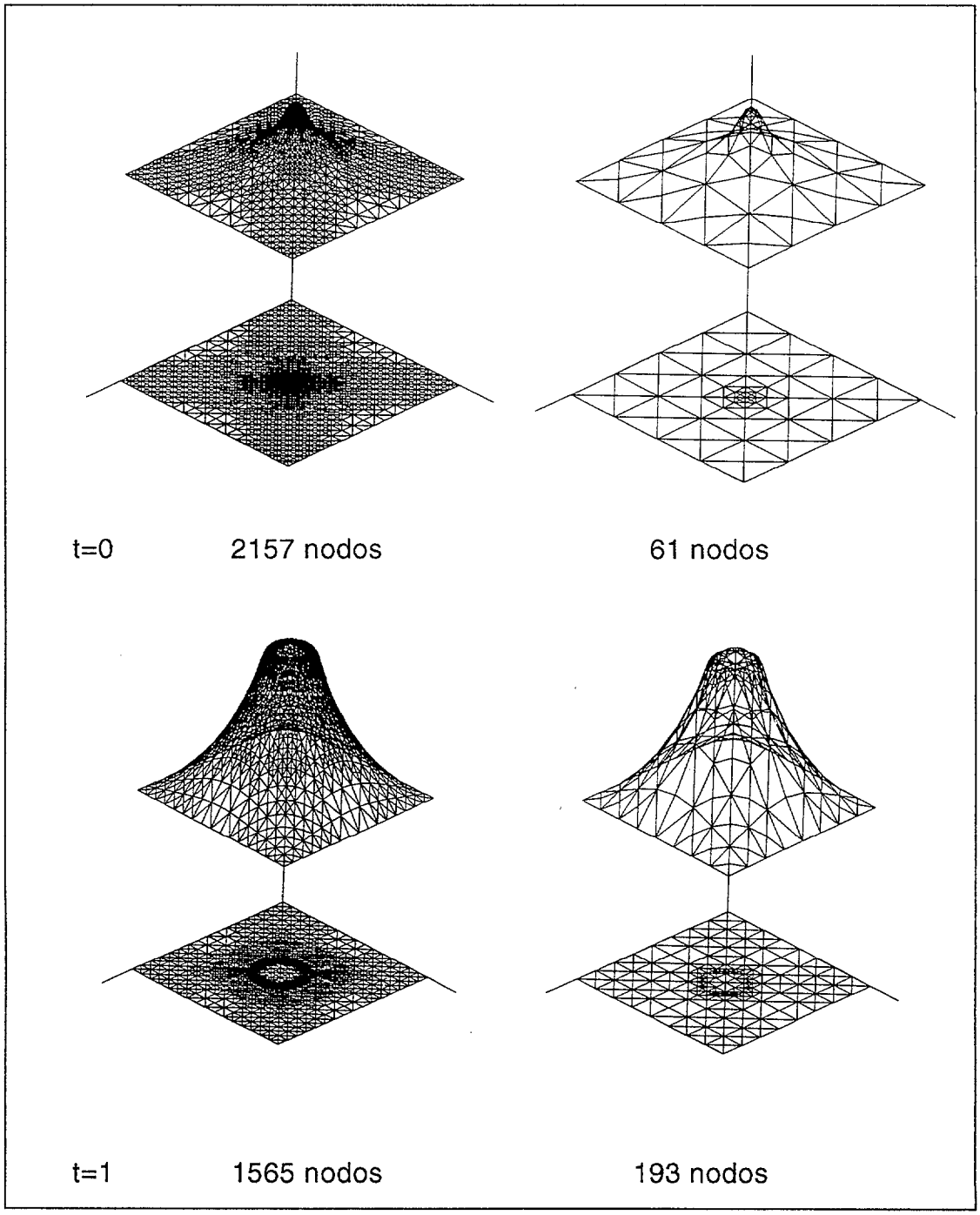
Como el parámetro t de la función del segundo miembro varía poco a poco, parece lógico que la solución correspondiente a valores cercanos de t sea también parecida. Por esta razón en lugar de partir, para cada nuevo valor del tiempo, de la malla inicial, realizaremos un desrefinamiento de la secuencia de mallas asociada a la solución del anterior valor del tiempo. Después de esto se resuelve el nuevo problema mediante la aplicación de varios refinamientos locales.

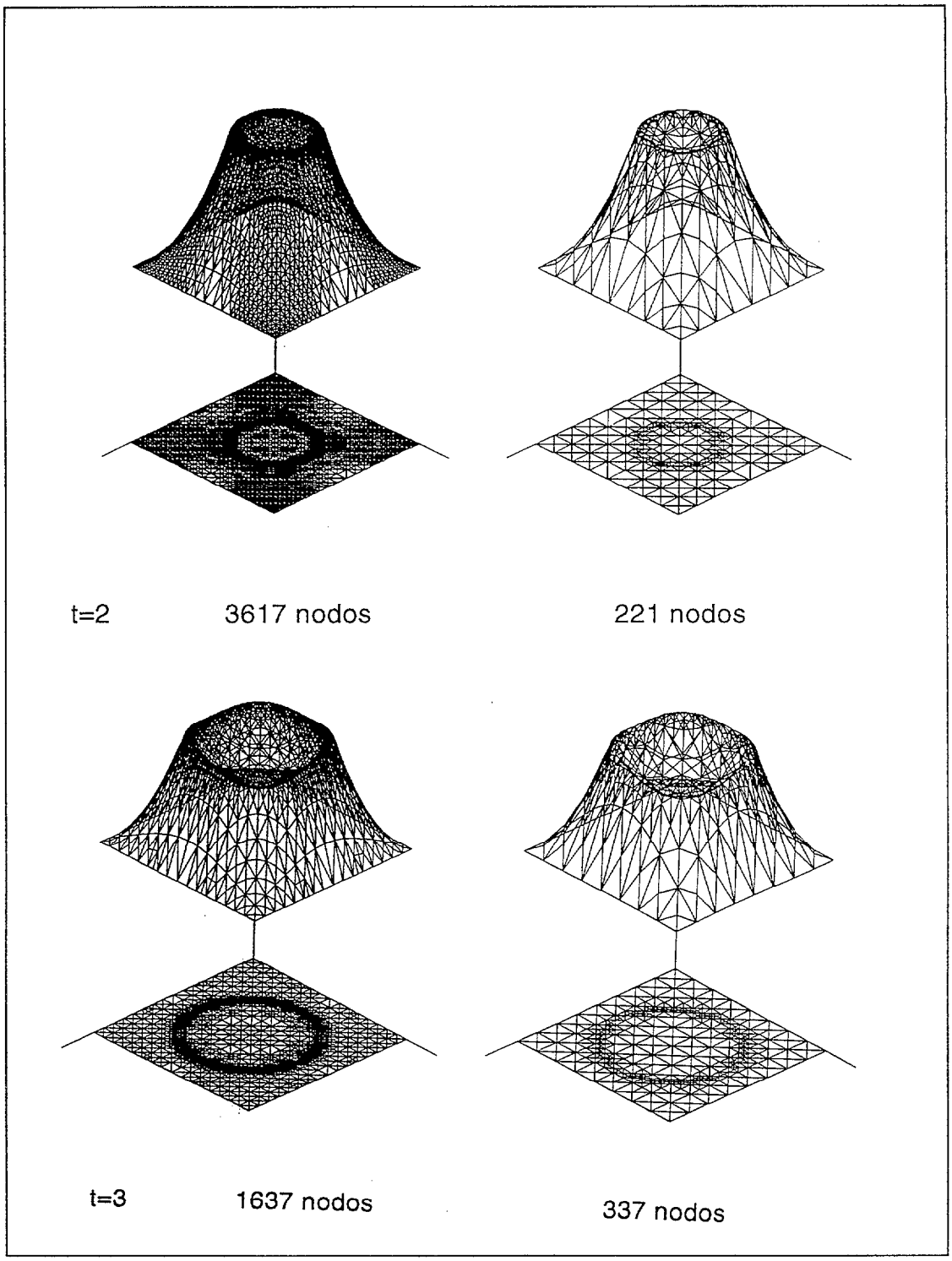
3.2.1. El problema modelo inicial

En las figuras de las páginas siguientes se observan los resultados gráficos al aplicar un desrefinamiento, con parámetro $\varepsilon=0.01$ y condición de desrefinamiento la diferencia relativa con el valor interpolado antes de resolver el problema para el nuevo valor de tiempo. Véase la diferencia de las mallas con respecto a las que aparecían tratando el problema solamente con refinamiento local (introducción). Se puede observar también cómo la malla desrefinada aproxima perfectamente la solución y cómo la combinación refinamientos-desrefinamiento hace que el número de nodos de las mallas refinadas permanezcan en un pequeño intervalo, en este ejemplo incluso disminuyen.

Además se puede resaltar en este problema también el gran ahorro de tiempo de cálculo. Para resolver el mismo problema en los primeros cuatro pasos de tiempo el tiempo de cálculo es 180 veces menor utilizando el algoritmo de desrefinamiento que mediante la estrategia que se comentó en

la introducción. Puede objetarse que aquella estrategia no era la mejor en el sentido de que se hacían 5 refinamientos locales con un parámetro $\gamma = 0.6$, y tal vez con un parámetro mayor, o menor número de refinamientos por paso de tiempo obtendríamos más rápidamente la solución. De cualquier forma, es patente la mejora obtenida al introducir el desrefinamiento; ha sido sustancial el ahorro en tiempo de calculo, al disminuirse drásticamente los números de ecuaciones.





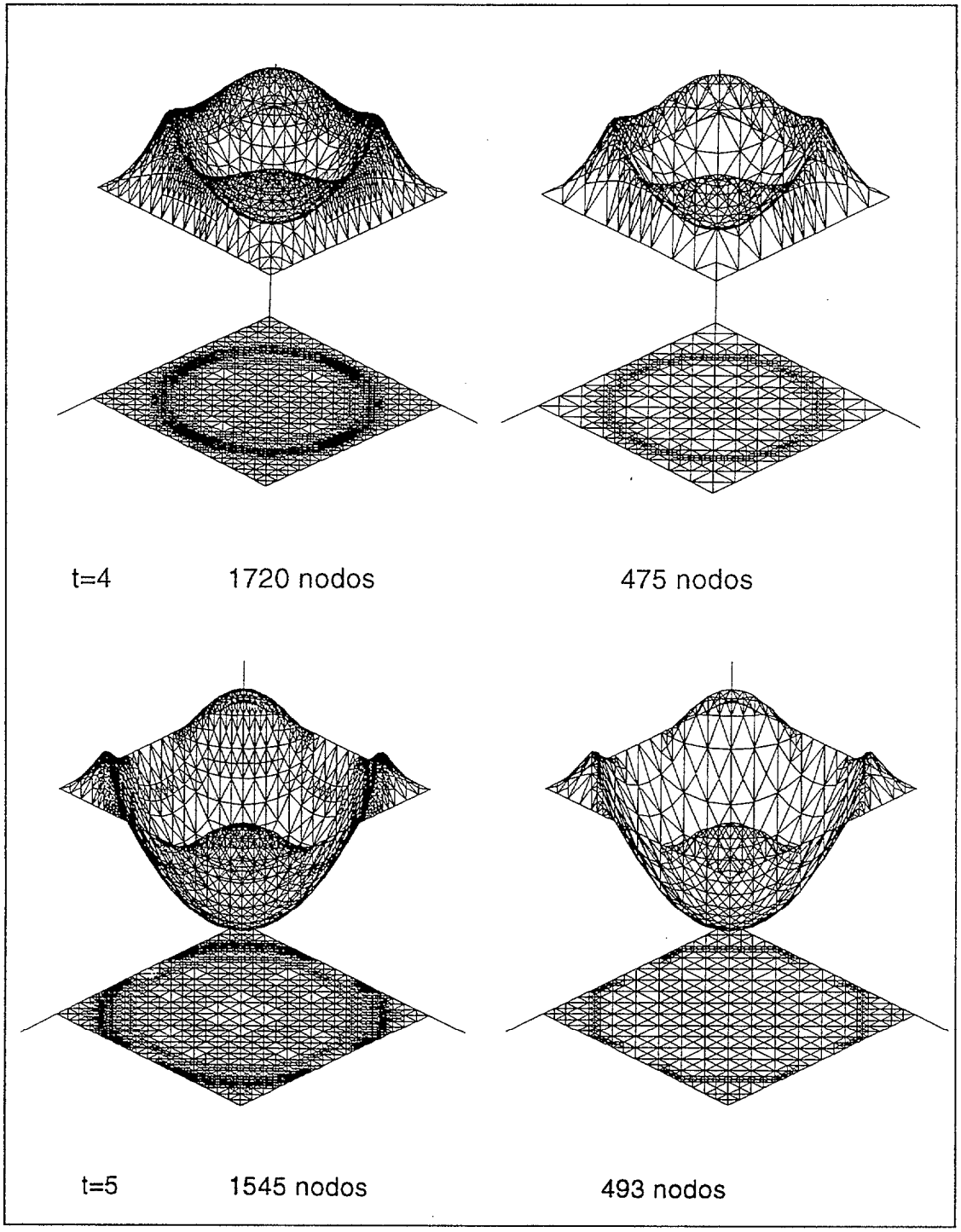


Figura 41.- A la izquierda mallas y solución en 3 dimensiones para sucesivos valores de tiempo; a la derecha una vez desrefinadas.

3.2.2. Un problema en geometría no regular

En este segundo ejemplo se resuelve otro problema de Poisson. El dominio, de geometría irregular, y las condiciones de contorno se muestran en la fig. 42. El problema es quasi-evolutivo, porque, como en el anterior, la función *f* del segundo miembro depende del tiempo según la expresión:

$$f(x,y)=\begin{cases}40 & si \quad d(t)<r\\ 0 & si \quad d(t)>r\end{cases}$$

donde *r* es el radio de la fuente circular (sombreada en la figura) y *d(t)* es la distancia euclídea entre un punto del dominio *(x,y)* y el centro de la fuente que se mueve con el tiempo con velocidad angular $\omega=\frac{\pi}{4} rad/seg$ en sentido anti-horario.

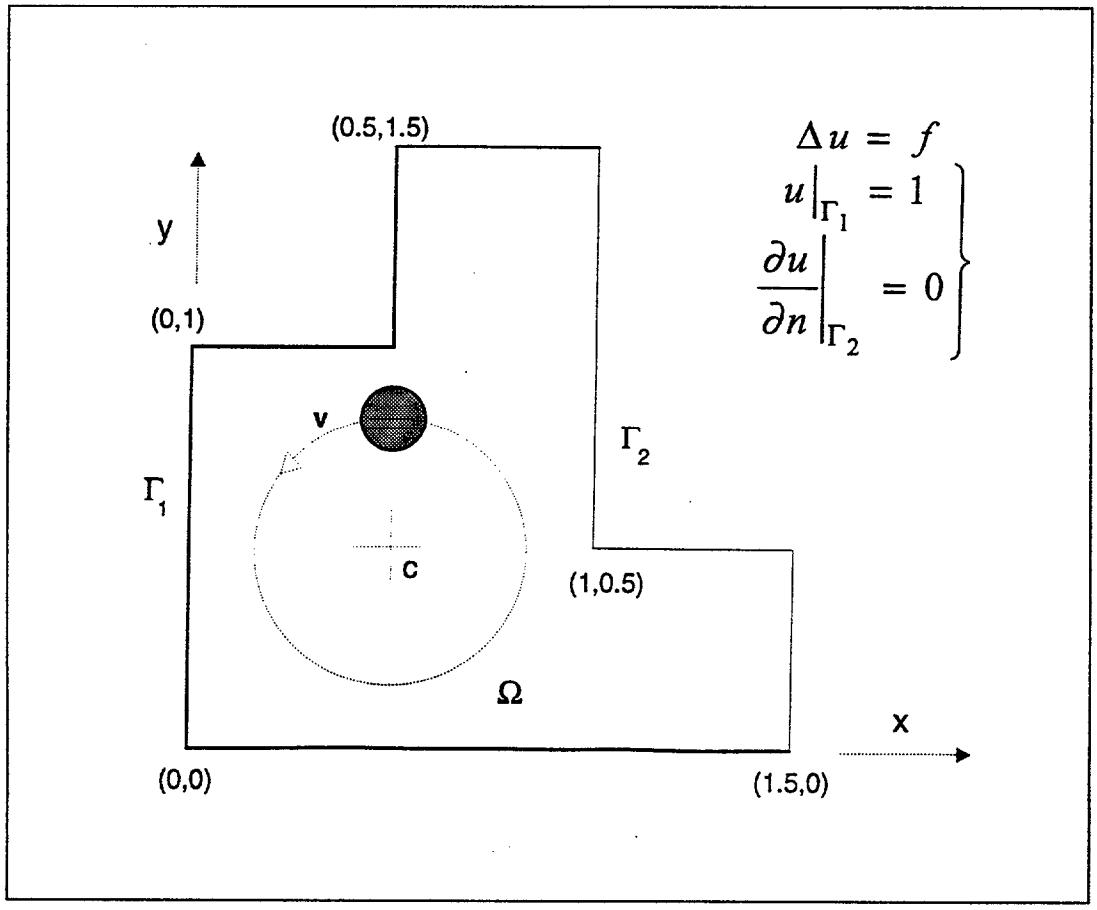
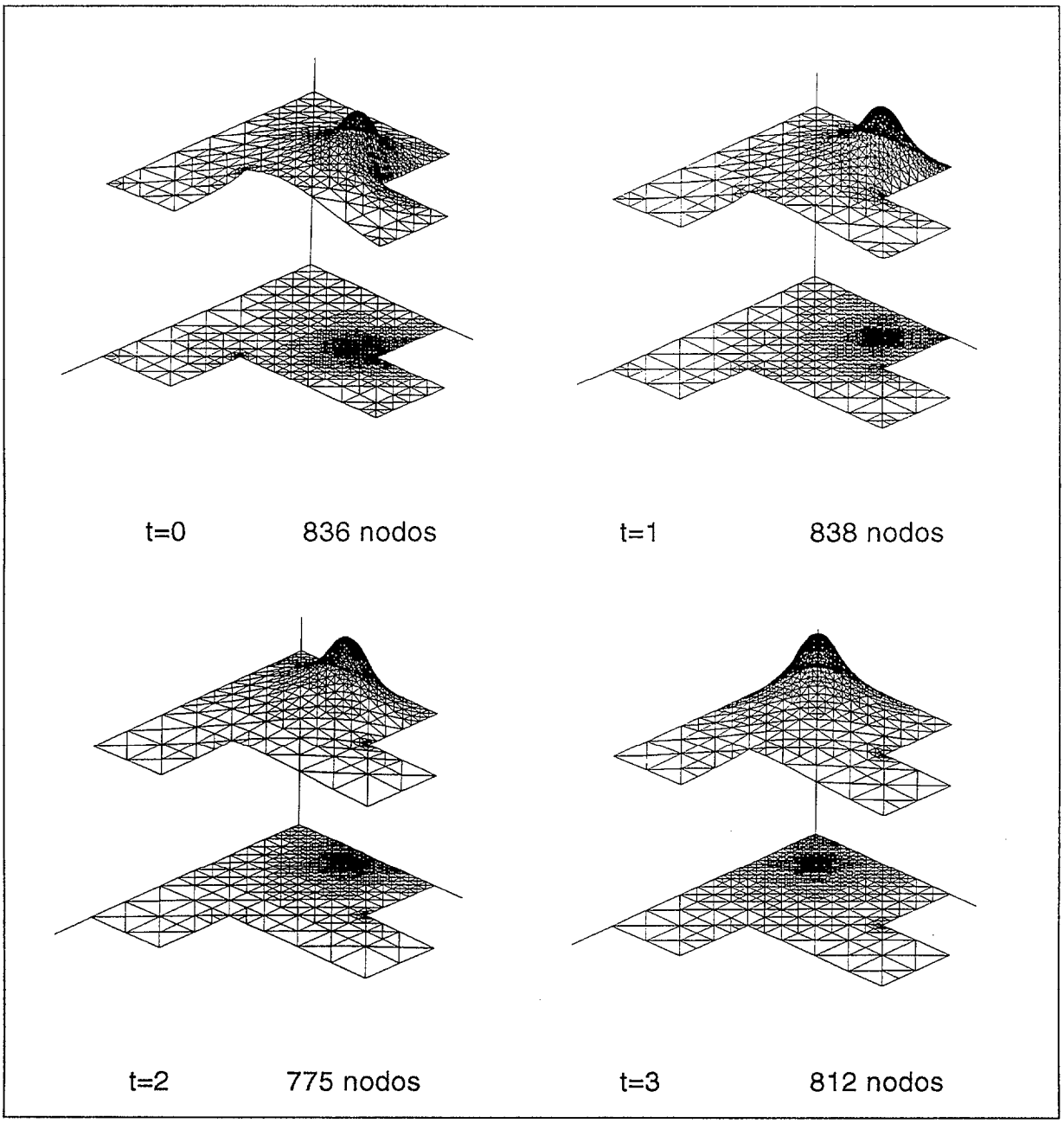


Figura 42.- Dominio y condiciones de contorno. Problema en geometría irregular.

En este ejemplo se ha utilizado el desrefinamiento como refinamiento local. La estrategia utilizada fue: un refinamiento global seguido de un desrefinamiento con parámetro de desrefinamiento $\varepsilon = 0.009$ y diferencia absoluta entre la solución numérica y la solución interpolada como condición de desrefinamiento. Desrefinar después de un refinamiento global equivale a un refinamiento local. Para aproximar la solución en cada paso de tiempo se empleo tres veces un refinamiento seguido de un desrefinamiento. En la figura se muestran algunas mallas y las correspondientes soluciones para diferentes valores de tiempo.



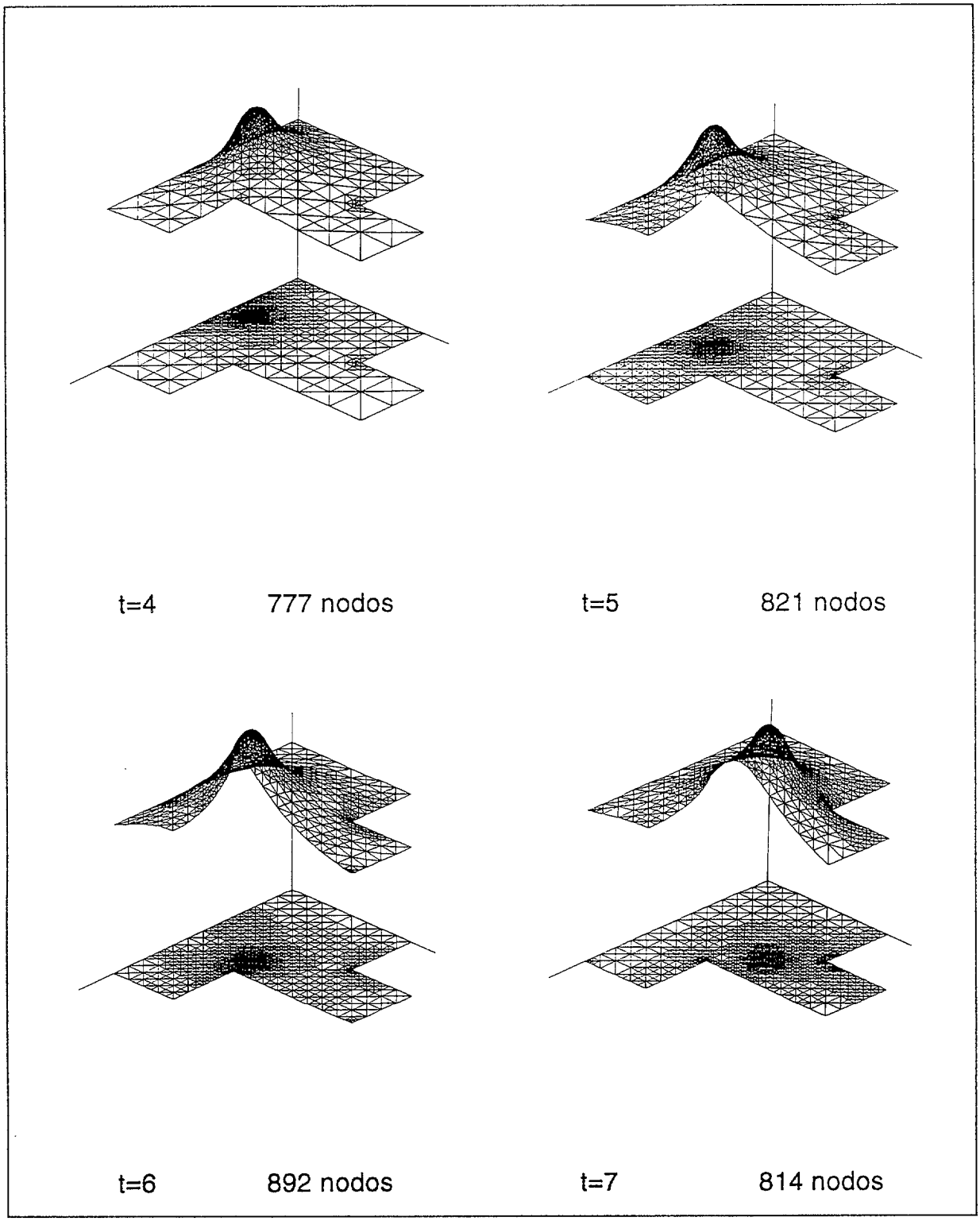


Figura 43.- En esta página y la anterior, diferentes mallas y soluciones para un problema de geometría no regular.

En cuanto a la resolución del sistema de ecuaciones asociado, se ha empleado un método multimalla con gradiente conjugado para el suavizado. Se ha alcanzado la convergencia con un valor estabilizado alrededor de cuatro iteraciones.

3.3. El problema de convección-difusión

Antes de presentar algunos resultados numéricos relativos a un problema de convección-difusión, se resumen en los siguientes apartados los principales aspectos numéricos sobre la formulación semi-implícita del problema. Esta formulación está estudiada con detalle en [Monte89] y [MoMoW89].

3.3.1. Formulación semi-implícita

Consideremos el problema de convección-difusión definido en un dominio bidimensional Ω, de frontera Γ:

$$\frac{\partial u}{\partial t}+\vec{v}\cdot\vec{\nabla}u-\vec{\nabla}\cdot\left(k\vec{\nabla}u\right)=f \qquad (1)$$

donde $u=u(\vec{x},t)$ es la solución del problema, que puede ser, por ejemplo, la temperatura o la concentración de un elemento del fluido en movimiento; ésta depende de su vector de posición, $\vec{x}=x_1\vec{i}+x_2\vec{j}$, y del tiempo t. El fluido que transporta la magnitud, u, posee un campo de velocidades en régimen estacionario dado por $\vec{v}=\vec{v}(\vec{x})$. Se ha estudiado el modelo lineal, donde la difusión viene dada por $k=k(\vec{x})$; y $f=f(\vec{x},t)$ son las fuentes externas. Se suponen condiciones de contorno e iniciales que aseguren la existencia y unicidad de la solución.

La ecuación (1) se puede escribir en términos de la derivada total, como:

$$\frac{du}{dt}-\vec{\nabla}\cdot\left(k\vec{\nabla}u\right)=f \qquad (2)$$

Un elemento del fluido que se encuentra en un instante t_n, en un punto $\underline{P}$, definido por $\underline{\vec{x}}$; al trasladarse con velocidad $\vec{v}$, después de un tiempo Δt,

pasará a la posición P, definida por $\vec{x}$, en el instante de tiempo t_{n+1}. Si hacemos $\vec{x} = \underline{\vec{x}} + \Delta\vec{x}$, y usamos la aproximación:

$$\frac{du}{dt} \approx \frac{u(\vec{x},t_{n+1}) - u(\underline{\vec{x}},t_n)}{\Delta t} = \frac{u^{n+1}(\vec{x}) - u^n(\underline{\vec{x}})}{\Delta t} \tag{3}$$

entonces, aplicando un esquema de Euler implícito a la ecuación (2), se obtiene:

$$u^{n+1}(\vec{x}) - \Delta t\, \vec{\nabla} \cdot \left[k(\vec{x}) \vec{\nabla} u^{n+1}(\vec{x}) \right] = \Delta t\, f^{n+1}(\vec{x}) + u^n(\underline{\vec{x}}) \tag{4}$$

Con el fin de poder evaluar $u^n(\underline{\vec{x}})$, escribiremos (4) en función de lo que ocurre en el punto P en cada instante de tiempo. Por esto se considera el desarrollo de Taylor de $\underline{\vec{x}}$. Para $i = 1, 2$:

$$\underline{x}_i = x_i(t_{n+1} - \Delta t) = x_i - v_i(\vec{x})\Delta t + \frac{\Delta t^2}{2} \vec{v}(\vec{x}) \cdot \vec{\nabla} v_i(\vec{x}) + 0(\Delta t^3)$$

De esta ecuación se obtiene $\Delta x_i = x_i - \underline{x}_i$, que se usará en el desarrollo de $u^n(\underline{\vec{x}})$:

$$u^n(\underline{\vec{x}}) = u^n(\vec{x} - \Delta\vec{x}) = u^n(\vec{x}) - \sum_{i=1}^{2} \Delta x_i \frac{\partial u^n(\vec{x})}{\partial x_i} + \\ + \frac{1}{2}\left[2\Delta x_1 \Delta x_2 \frac{\partial^2 u^n(\vec{x})}{\partial x_1 \partial x_2} + \sum_{i=1}^{2} \Delta x_i^2 \frac{\partial^2 u^n(\vec{x})}{\partial x_i^2} \right] + 0(\|\Delta\vec{x}\|^3)$$

y entonces,

$$u^n(\underline{\vec{x}}) = u^n(\vec{x}) - \Delta t \sum_{i=1}^{2} v_i(\vec{x}) \frac{\partial u^n(\vec{x})}{\partial x_i} + \frac{\Delta t^2}{2} \sum_{i=1}^{2} \left[\vec{v}(\vec{x}) \cdot \vec{\nabla} v_i(\vec{x}) \right] + \\ + \frac{\Delta t^2}{2} \sum_{i=1}^{2} \sum_{j=1}^{2} v_i(\vec{x}) v_j(\vec{x}) \frac{\partial^2 u^n(\vec{x})}{\partial x_i \partial x_j} + 0(\Delta t^3)$$

Finalmente, si introducimos esta última expresión en (4), se obtiene la siguiente formulación semi-implícita que aproxima el proceso de convección-

difusión y donde todos los términos se han evaluado en el mismo punto P, definido por $\vec{x}$:

$$u^{n+1} - \Delta t\, \vec{\nabla} \cdot \left[k \vec{\nabla} u^{n+1} \right] = \Delta t\, f^{n+1} + u^n - \Delta t\, \vec{v} \cdot \vec{\nabla} u^n + \\ + \frac{\Delta t^2}{2} \sum_i (\vec{v} \cdot \vec{\nabla} v_i) \frac{\partial u^n}{\partial x_i} + \frac{\Delta t^2}{2} \sum_{i,j} v_i v_j \frac{\partial^2 u^n}{\partial x_i \partial x_j} \qquad (5)$$

Ahora si consideramos las condiciones de contorno, es fácil obtener la formulación variacional y aplicar entonces el método de los elementos finitos. Se ha usado integración consistente en todos los términos de la formulación final.

3.3.2. Estabilidad y consistencia

Con respecto a la estabilidad se ha hecho el estudio en el sentido de Von Neumann. Estos resultados han sido analizados en [MoMoW90]. Allí se generalizan condiciones de estabilidad partiendo del estudio en una dimensión sobre una malla regular.

Sean los números de *Courant* y *Peclet*:

$$\begin{cases} C = \dfrac{v\,\Delta t}{h} \\ Pe = \dfrac{v\,h}{K} \end{cases}$$

entonces, en una dimensión se obtiene la siguiente condición de estabilidad:

$$C \le \left[\frac{1}{P_e^2} + \frac{1}{3} \right]^{1/2} + \frac{1}{P_e} \qquad (6)$$

En la referencia citada no se hace un estudio general de la estabilidad en dos dimensiones. Sin embargo, sí se estudia el caso particular de una malla de triángulos rectángulos isósceles cuyas hipotenusas están orientadas siguiendo un ángulo de $\pi/4$ radianes y sus longitudes son $h\sqrt{2}$. Se supone un campo de velocidades con dirección definida por un ángulo α con el eje de abscisas. Sólo se estudian dos direcciones de propagación de ondas: $\alpha = 0$ y $\alpha = 3\pi/4$. En esos casos se obtienen, respectivamente, las siguientes cotas de estabilidad:

$$C \le \frac{1}{\cos^2 \alpha} \left[\left(\frac{1}{P_e^2} + \frac{1}{3} \cos^2 \alpha \right)^{1/2} + \frac{1}{P_e} \right] \qquad (7)$$

$$C \le \frac{1}{1 - \operatorname{sen} 2\alpha} \left[\left(\frac{4}{P_e^2} + \frac{1}{3} (1 - \operatorname{sen} 2\alpha) \right)^{1/2} + \frac{2}{P_e} \right] \qquad (8)$$

En ambos casos el caso más desfavorable resulta cuando las direcciones del campo de velocidades y de la propagación de ondas coinciden: $\alpha = 0$ en (7) y $\alpha = 3\pi/4$ en (8). En este supuesto se observa que la condición dada en (8) resulta peor que la (7). Además se puede comprobar que en el caso de que el campo de velocidades y la propagación de onda tengan la misma dirección, las dos últimas condiciones se pueden obtener de la (6) sin más que tomar *h* como la distancia máxima entre dos puntos del elemento bidimensional en esa dirección común.

En cuanto a la consistencia, se recuerdan resultados que pueden encontrarse en [PeZiM86] y [MoMoW90]. Se comparan, después de un paso de tiempo, la ganancia de la solución numérica y de la solución exacta. Dada la formulación semi-implícita analizada y particularizando en una dimensión y con coeficientes constantes sin fuerzas externas, se puede probar que la aproximación es globalmente de segundo orden, es decir que:

$$G(\eta) = G_e(\eta) + O(\eta^3)$$

donde $\eta = \xi h$, siendo ξ el número de onda; y $G(\eta)$ y $G_e(\eta)$ las ganancias de después de un paso de tiempo en un nodo genérico de una malla regular, de la solución numérica y de la solución exacta respectivamente.

3.3.3. Estrategia adaptativa

En [Monte89] se ha resuelto el problema de convección-difusión usando diferentes indicadores de refinamiento para el refinamiento local. Sin embargo, nosotros, aunque hemos utilizado también los indicadores señalados, hemos introducido un nuevo indicador, ε_i, para un elemento Ω_i:

$$\varepsilon_i = h_i^2 \left| \nabla u_h \right|$$

siendo h_i el diámetro del elemento Ω_i y u_h la solución numérica lineal en el elemento triangular. Este indicador de refinamiento es una medida de la máxima variación de u_h en Ω_i ponderada con el diámetro h_i del elemento, con el fin de que las áreas de alto gradiente no se refinen excesivamente. Con este indicador se logran regiones de refinamiento no tan localizadas como con el indicador:

$$\varepsilon_i = h_i \left| \nabla u_h \right|$$

3.3.4. Resultados numéricos

Consideremos el siguiente problema modelo con objeto de mostrar la eficacia de los procesos readaptativos en los problemas evolutivos. Sea un cuadrado de lado unidad, Ω, centrado en el punto $(0.5, 0.5)$, con condiciones de tipo Neumann nulas en su frontera Γ. Supongamos que en ese dominio tenemos un campo de velocidades giratorio $\vec{v} = (v_1, v_2)$, ver fig. 44, donde:

$$v_1 = \omega\, x_1(1-x_1)(x_2-0.5)$$
$$v_2 = \omega\, x_2(0.5-x_1)(1-x_2)$$

de tal forma que $\vec{\nabla}\cdot\vec{v} = 0$ y $\vec{v}$ es tangente a Γ.

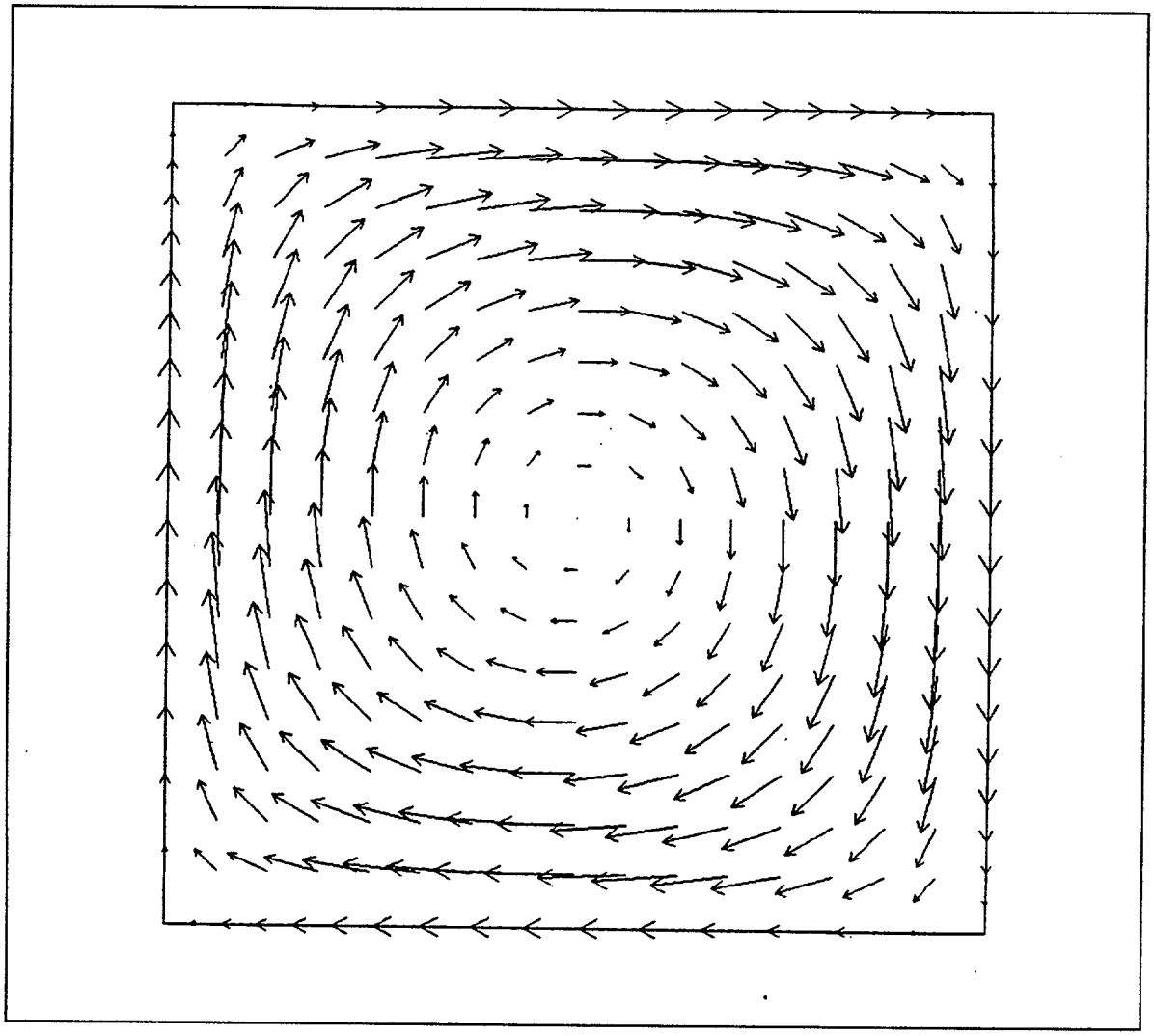


Figura 44.- Campo de velocidades.

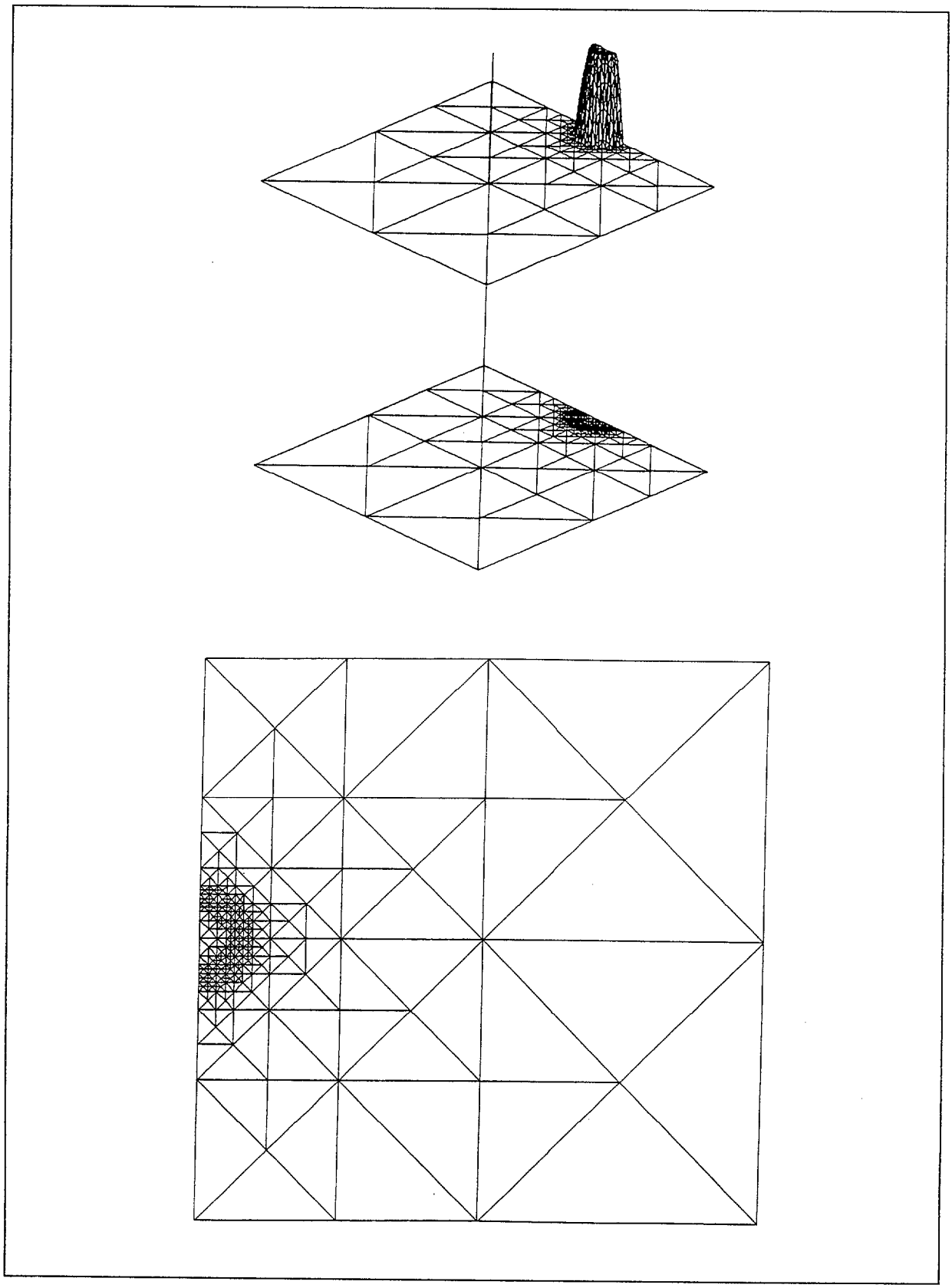
El máximo valor del módulo de la velocidad es $\frac{\omega}{8}$ y éste se alcanza en los puntos medios de los lados, mientras que en las esquinas del cuadrado la velocidad se hace cero. En la presente aplicación hemos elegido $\omega = 2.5 \cdot 10^4$. Se ha tomado como valor constante de la difusión $k = 1$. Por tanto, el valor del número de Peclet para este problema es $3.125 \cdot 10^3$. Se han supuesto también fuerzas externas nulas: $f = 0$.

La solución inicial tiene la forma de una meseta semicilíndrica como puede apreciarse en la fig. 45(a) y ha sido aproximada usando una estrategia readaptativa. Partiendo de una malla inicial muy grosera, compuesta por 8 triángulos y 9 nodos, se realizan 3 refinamientos globales para capturar la solución inicial. Después se ha aplicado un desrefinamiento con parámetro $\varepsilon = 0.005$ y diferencia absoluta para disminuir el número de nodos en el dominio. Este proceso se ha repetido dos veces, pasándose de 1081 nodos en la última malla refinada a 234 nodos en la malla desrefinada (fig. 45(a)). Mediante esta estrategia logramos una buena aproximación con un número de nodos mínimo. En concreto la solución inicial de este problema corresponde a la siguiente fórmula:

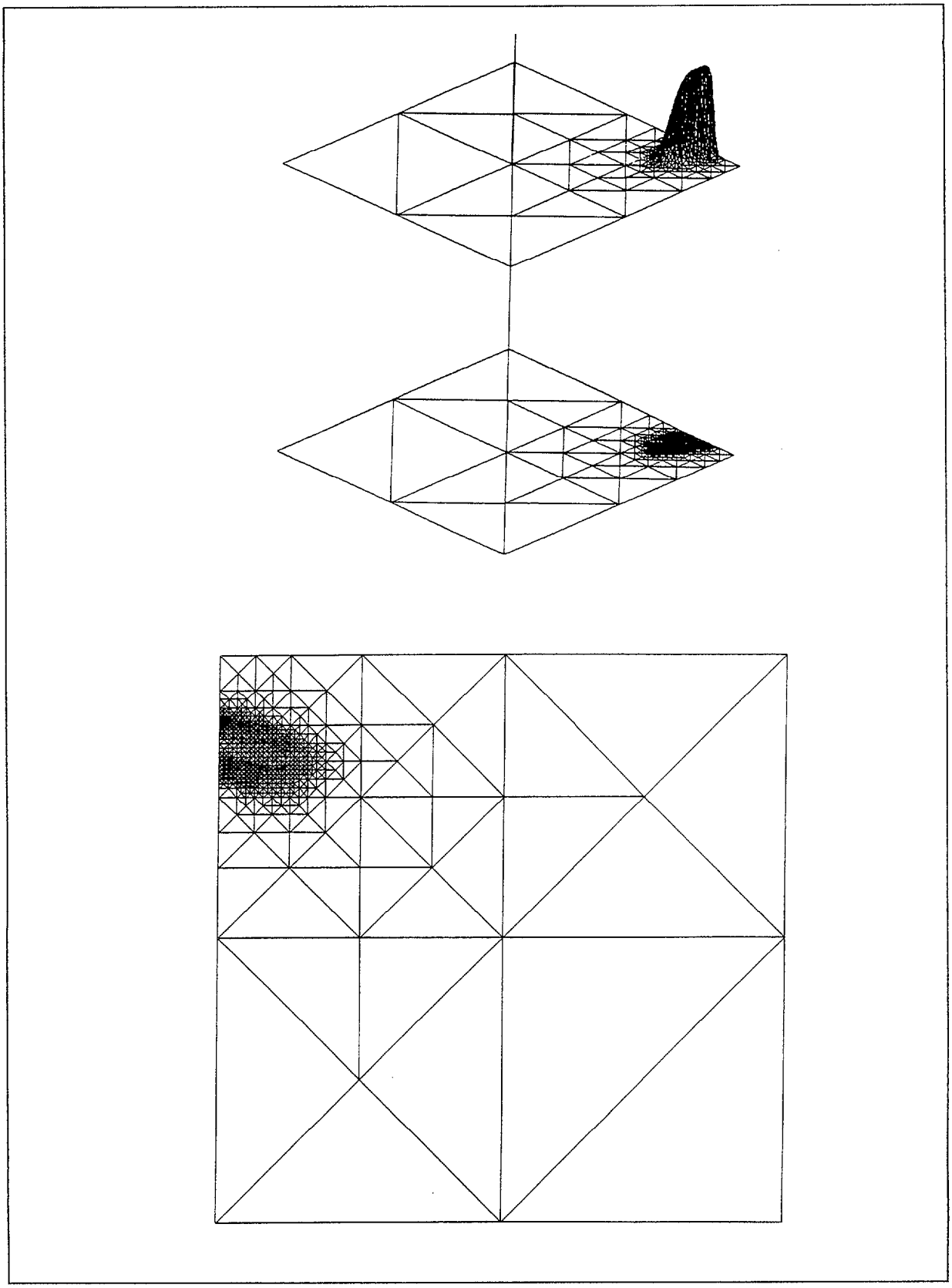
$$u_0(\vec{x}) = \begin{cases} 1 & \text{si} \quad d(\vec{x}) \geq 0.10 \\ 1 + 0.3 \cdot exp\left[\left(0.005 - d(\vec{x})\right) / \left(0.1 - d(\vec{x})\right)\right] & \text{si} \quad 0.05 < d(\vec{x}) < 0.1 \\ 1.3 & \text{si} \quad d(\vec{x}) \leq 0.05 \end{cases}$$

donde $d(\vec{x})$ representa la distancia euclídea de un punto del dominio $\vec{x}$ al centro de la meseta: $(0.0, 0.5)$. Se ha utilizado la función exponencial que aparece en la fórmula para *suavizar* el desnivel entre el máximo y el mínimo de la solución.

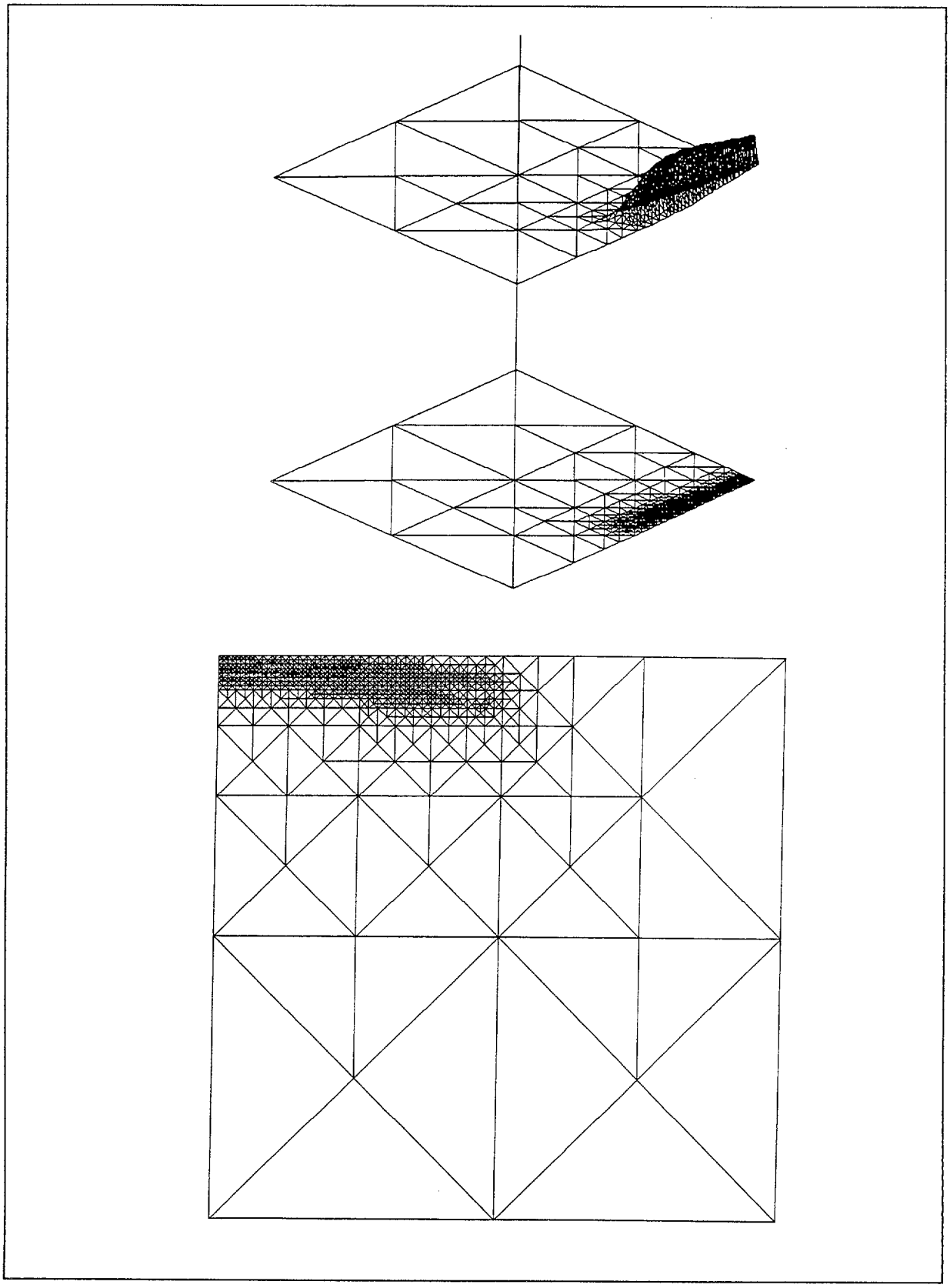
En las siguientes páginas se muestran las soluciones para distintos valores del tiempo. Aparecen también las mallas correspondientes a cada solución. Se aprecia cómo se adapta la malla a la solución, cambiante con el tiempo.



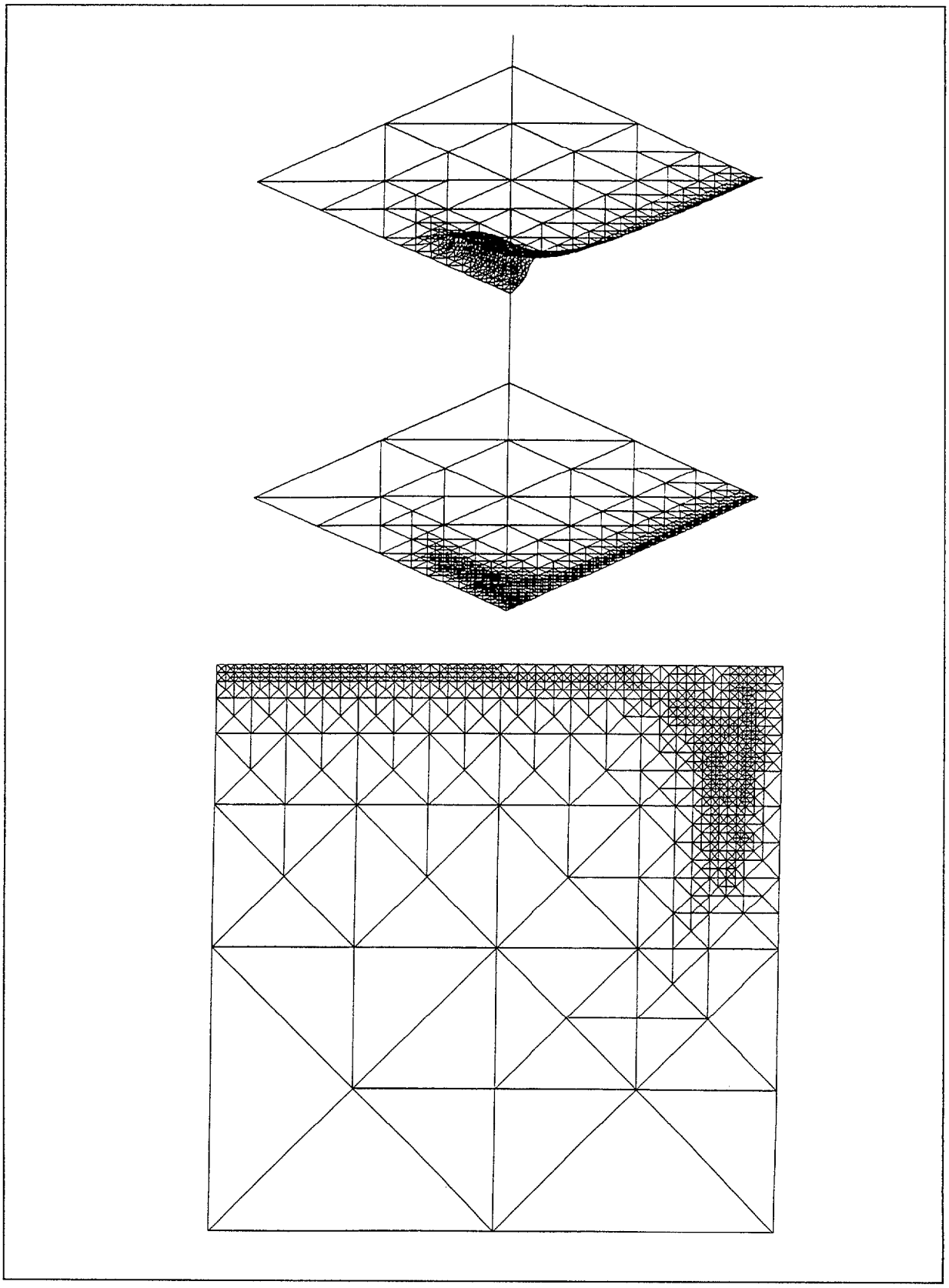
(a) Solución inicial, t = 0.0, 234 nodos.



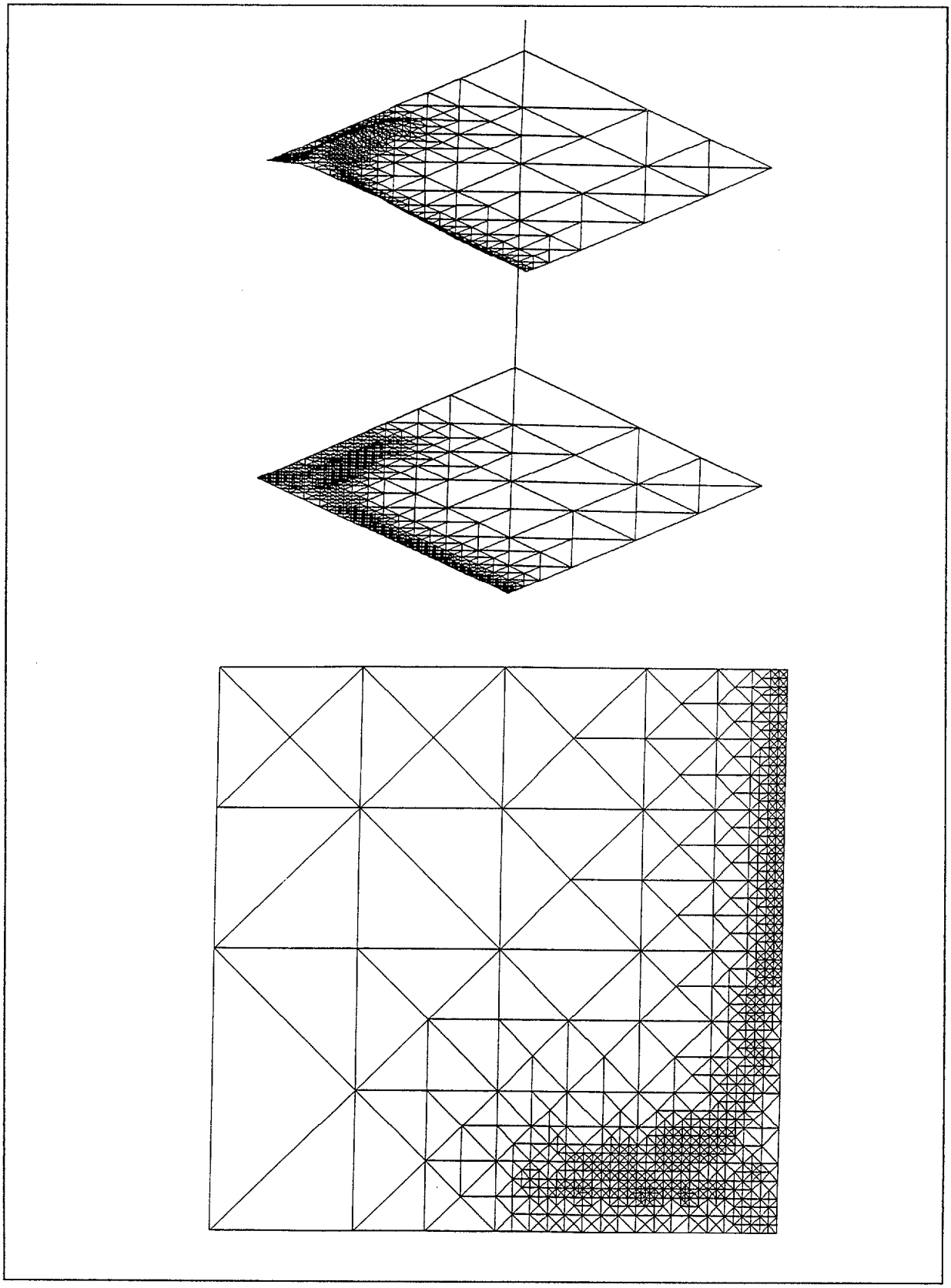
(b) t = 0.00014, 580 nodos.



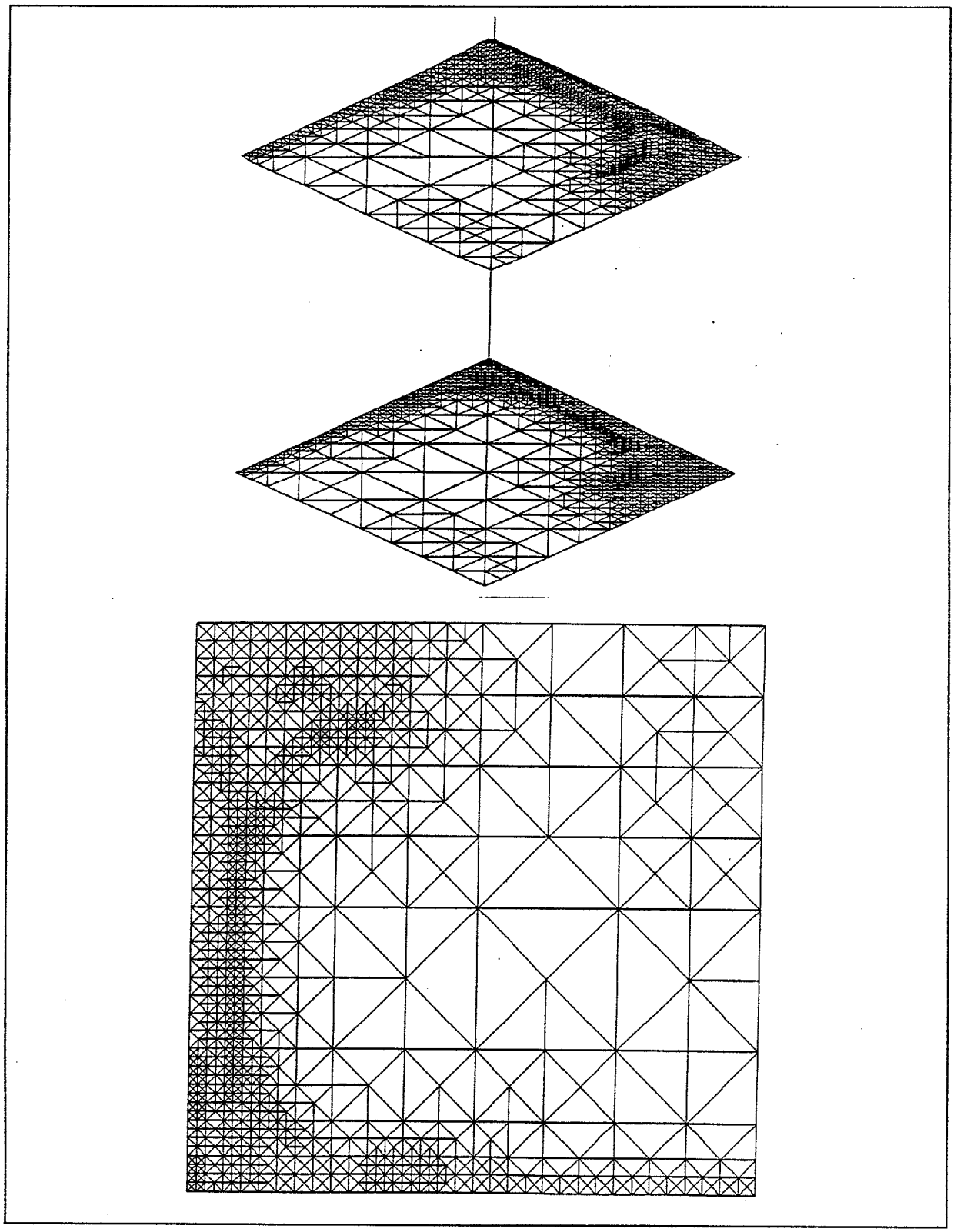
(c) t = 0.00032, 801 nodos.



(d) t = 0.00060, 628 nodos.



(e) t = 0.00097, 337 nodos.



(f) t = 0.0163, 221 nodos.

Figura 45 .- Mallas y soluciones para un problema evolutivo.

Se ha usado el indicador de error comentado en la sección anterior a la hora de refinar con parámetro $\gamma = 0.6$, y se ha aplicado un parámetro de desrefinamiento $\varepsilon = 0.001$, y condición de desrefinamiento la diferencia absoluta entre la solución numérica y la solución interpolada.

Es importante destacar, en este ejemplo, que el número de nodos y su situación en el mallado está controlado automáticamente por el código de acuerdo con la estrategia readaptativa utilizada y con la solución numérica en cada instante de tiempo.

Para llegar a la solución estacionaria, constante en este problema, como puede apreciarse en la figura, fueron necesarios realizar 2.892 pasos de tiempo. Se realizaron 723 desrefinamientos y 2.169 refinamientos locales. El máximo número de niveles de malla presentes en una de las secuencias manejadas en el proceso fue de 167. La evolución del número de niveles de malla respecto del número de desrefinamientos realizados se puede apreciar en la fig. 46.

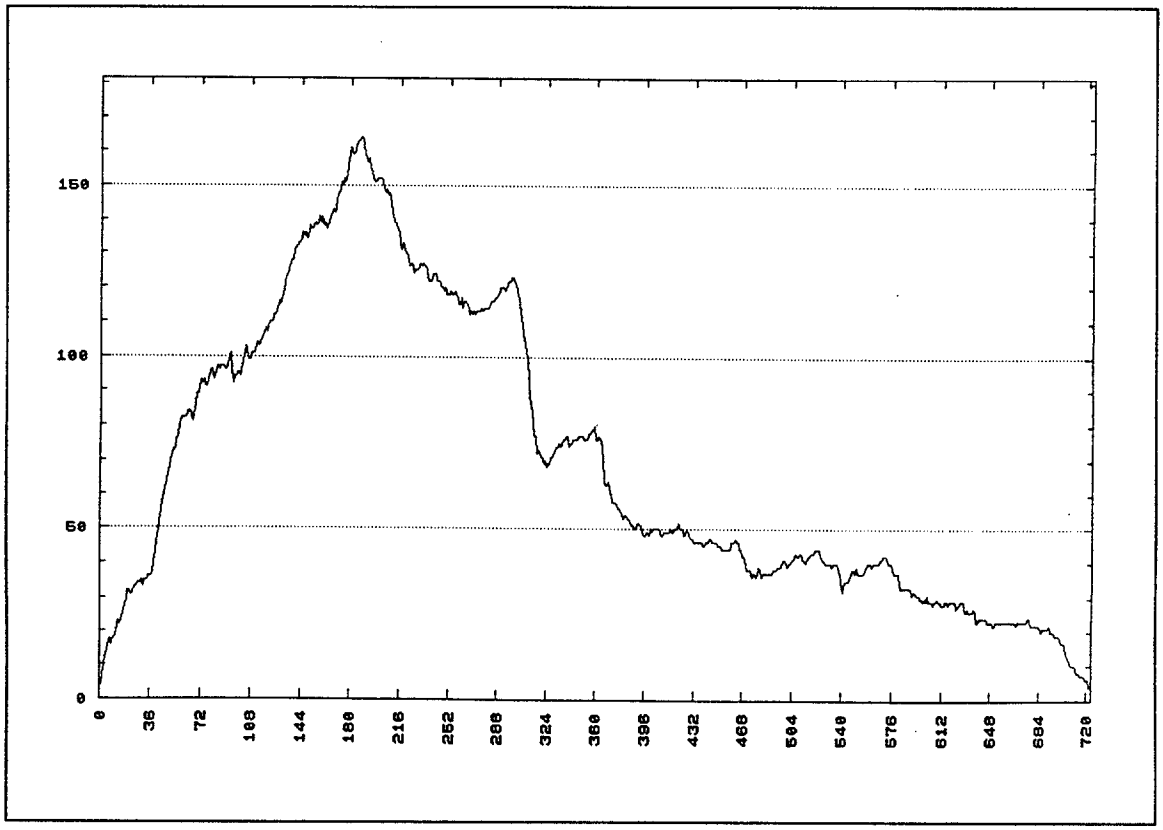


Figura 46.- Evolución del número de niveles.

La ejecución del programa se paró automáticamente al alcanzarse un número de niveles de malla que fueron definidos en la entrada de datos, en este caso fueron 3 niveles de malla, es decir, cuando quedaban 81 nodos igualmente distribuidos en el dominio, tras 6 horas y 20 minutos de CPU.

También se ha de resaltar el que para estudiar problemas evolutivos, en los que se conozca a priori cómo es la solución estacionaria se puede aplicar una estrategia semejante. Esto tiene la ventaja de que no es necesario estimar *a priori* el número de pasos de tiempo que hay que realizar. Además, si la solución estacionaria no es conocida, se pueden dar también condiciones sobre la evolución del número de nodos (o de niveles de malla) que permitan abortar la ejecución de un programa, si se ha alcanzado la solución estacionaria.

La evolución del número de nodos en todo el proceso evolutivo se observa en la fig. 47.

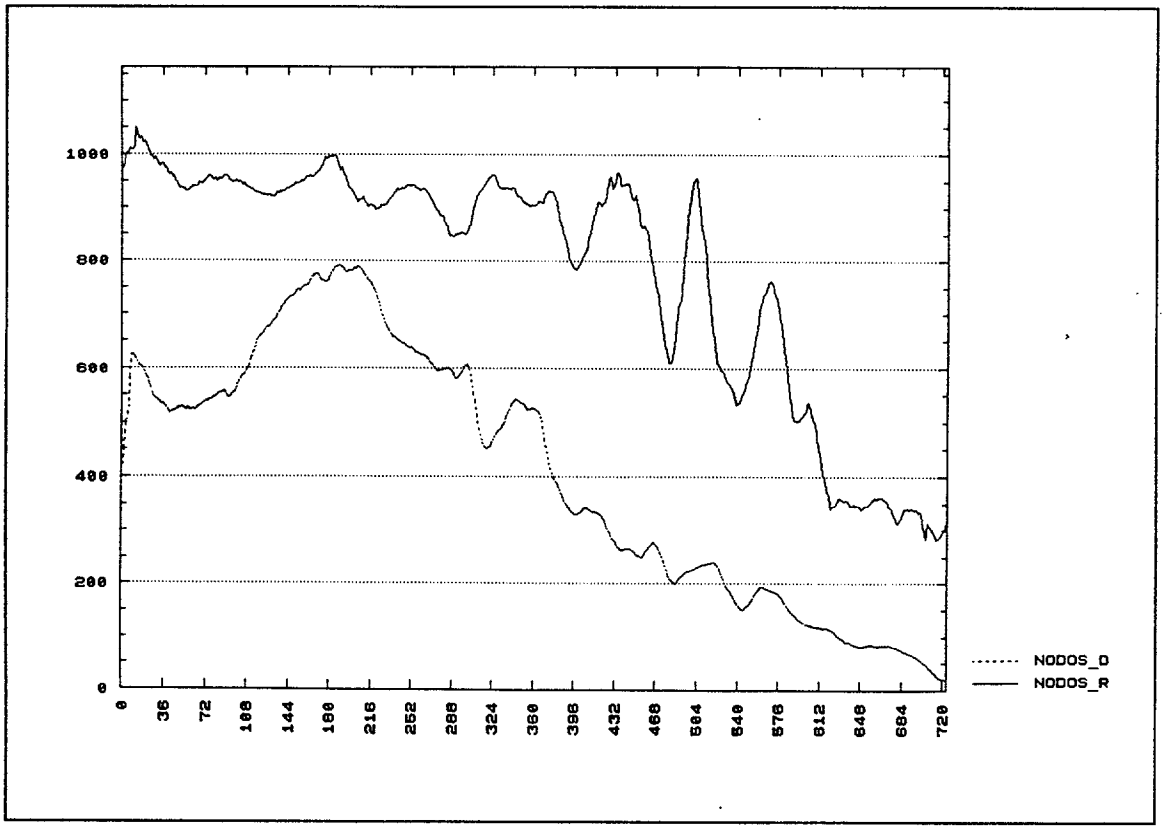


Figura 47.- Evolución del número de nodos respecto número de desrefinamientos.

Es evidente que el número de nodos está acotado en todo el proceso. En este caso tiende a disminuir a medida que se difunde la solución inicial.

También se puede señalar que la relación de nodos por nivel de malla varía en el proceso. Esta puede apreciarse en la fig. 48. Tanto al principio como al final esos valores son muy altos porque se realizan refinamientos prácticamente globales. Por este motivo se han suprimido de la figura. Se puede observar que en las mallas desrefinadas esta relación es menor de 10 en gran parte del problema. En las mallas refinadas hay más oscilaciones, debido a que el parámetro de refinamiento sólo indica que los elementos que tengan un error local superior a 0.6 veces el error máximo han de ser refinados. Según sea la distribución de errores, un mismo parámetro de refinamiento puede hacer que sean introducidos más o menos nodos en el mallado.

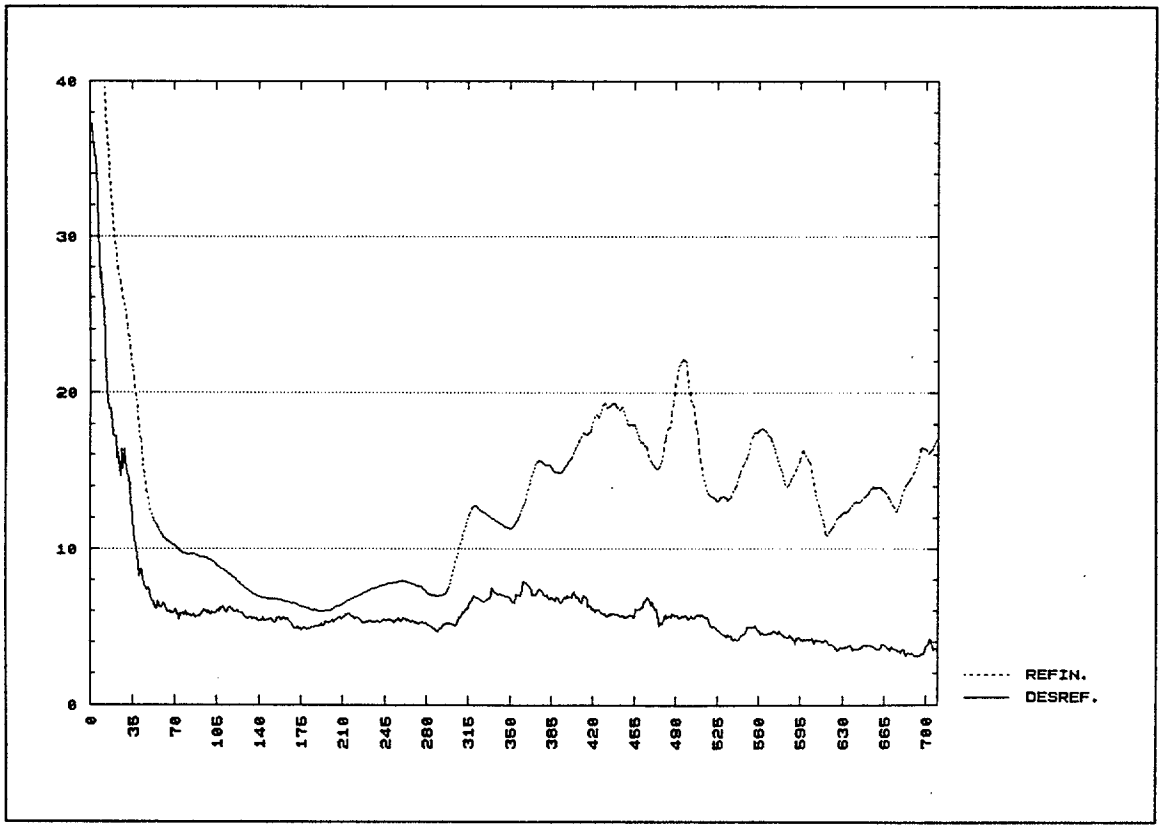


Figura 48.- Relación nº. nodos/nº. niveles.

Tanto por el gran número de niveles de malla involucrados en el proceso evolutivo, como por la poca diferencia de nodos entre niveles de malla consecutivos, se han efectuado suavizados o post-suavizados en niveles de malla no consecutivos en la aplicación del método multimalla, de manera que la relación de nodos entre niveles efectivos sea $n_\ell / n_{\ell-1} = 2.5$. Véase a este respecto el apartado 2.6.4. Se ha alcanzado la convergencia en una iteración multimalla.

Sin embargo, la utilización que se ha hecho aquí del método multimalla, es decir como herramienta para resolver en cada paso de tiempo el correspondiente sistema lineal de ecuaciones, no es la más efectiva en la práctica. En su lugar se podría utilizar un método multimalla que integrase también la variable temporal, por ejemplo en la línea propuesta recientemente en [PerPe93]. En el algoritmo allí descrito las soluciones correspondientes a las mallas más groseras son seguidas también en el tiempo, haciendo uso del hecho de que en ellas están permitidos pasos de tiempo mayores.

4

Conclusiones

Como conclusiones más importantes de este trabajo se pueden destacar las siguientes:

1.- El algoritmo de desrefinamiento, combinado con el refinamiento (local o global) es muy útil para resolver problemas dependientes del tiempo, en los cuales se requieren áreas de refinamiento móviles. El uso del refinamiento con parámetro variable permite que el propio código adapte la malla según el número de nodos considerado óptimo, y es un paso más en la progresiva automatización de un proceso de elementos finitos.

2.- El algoritmo de desrefinamiento desarrollado hace posible utilizar fácilmente el método multimalla para resolver el sistema de ecuaciones asociado al método de los elementos finitos. Esto hace que el método multimalla, junto con los mallados estructurados con procesos readaptativos sean particularmente útiles en problemas que involucran gran número de incógnitas.

3.- Sin la utilización del algoritmo de desrefinamiento el coste en tiempo de CPU para la resolución de problemas evolutivos con números de Peclet tan elevados como los propuestos en este trabajo, o, incluso en problemas quasi-evolutivos, puede hacer prohibitiva su resolución en el marco de los mallados estructurados.

4.- Se ha comprobado la formulación semi-implícita para el problema de convección-difusión propuesta en [MoMoW89] y en [Monte89].

5.- El algoritmo promete ser muy útil en problemas más complejos, incluidos problemas no lineales, en los cuales permitirá que la malla se

adapte a la solución cambiante con el tiempo, y se aprovechará más el bajo coste computacional de los métodos multimalla cuando hay un gran número de incógnitas.

6.- Se puede usar el algoritmo de desrefinamiento como un refinamiento local, incluso en problemas estacionarios. Desrefinar después de un refinamiento global es equivalente a un refinamiento local. En ese caso no se necesitan indicadores de error. Esto implica claras ventajas en aquellos problemas para los cuales no existen indicadores de error o su cálculo en demasiado costoso.

7.- En problemas estacionarios puede utilizarse el algoritmo de desrefinamiento como criterio de parada. Si al desrefinar se eliminan todos -o casi todos- los nodos propios de la malla fina se puede parar la ejecución del programa y almacenar la solución correspondiente a la malla refinada.

8.- El algoritmo se puede utilizar también para obtener el mejor soporte a trozos para interpolar linealmente una función de dos variables dada, o para aproximar la solución inicial de un problema evolutivo. Asimismo puede utilizarse para resolver el problema de encontrar una malla inicial de un dominio plano que sea capaz de representar alguna propiedad física o de contorno con un número mínimo de nodos.

5

Líneas futuras de investigación

El trabajo desarrollado sugiere algunas líneas de investigación entre las que se pueden señalar las siguientes:

1.- Además de la pequeña mejora del algoritmo descrito apuntada en el apartado 2.6.6. este trabajo sugiere el desarrollo de un algoritmo de refinamiento *estructurado* capaz de refinar la malla, añadiendo los nodos *tan pronto como sea posible*, es decir, en el nivel de malla más bajo de la secuencia de mallas encajadas. Así como el desrefinamiento maneja toda la secuencia de malla encajadas, el nuevo algoritmo de refinamiento del que hablamos debería manejar también toda la secuencia de mallas. Unas primeras ideas sobre este algoritmo aparecen en el apéndice *a1*.

2.- Se puede estudiar con más detalle la posibilidad de optimizar la estructura de datos del código de forma que se minimice el espacio requerido en memoria. Esto puede ser conveniente en problemas con gran número de nodos y de niveles de malla, o problemas en tres dimensiones.

3.- Generalización a tres dimensiones tanto de un algoritmo de refinamiento en mallados estructurados, como de su algoritmo de desrefinamiento inverso. En este sentido la estructura de datos actual del código Neptuno es fácilmente adaptable debido a su carácter jerárquico. Actualmente se está trabajando en el Departamento en el desarrollo de un mallador tridimensional y se han obtenido los primeros resultados positivos. Cabría preguntarse si es posible desarrollar un refinamiento en el marco de los mallados estructurados que conserve las buenas propiedades que tiene el 4-T de Rivara en dos dimensiones.

4.- La aplicación del algoritmo de desrefinamiento a una imagen bidimensional sugiere un amplio campo de aplicación del método de los

elementos finitos al tratamiento de imágenes: filtrado, restauración, segmentación de imágenes, discriminación de texturas, a través de la resolución de ecuaciones diferenciales en derivadas parciales no lineales por una parte, y a la compresión de imágenes por otra. Habría que dotar, en este último caso al algoritmo de desrefinamiento de una estructura de datos que lo hiciera competitivo y rápido, para que se pudiera utilizar en tiempo real.

5.- Otras futuras líneas de investigación son las aplicaciones de los procesos readaptativos a problemas evolutivos lineales y no lineales de todo tipo, en los que prometen ser muy eficaces.

Apéndices

a1

El refinamiento estructurado

En este apartado se indica una alternativa al algoritmo de refinamiento en mallados estructurados que, además, obvia el algoritmo de compactación de niveles del apartado 2.6.3. La idea es que no sería necesario eliminar niveles de malla si éstos no crecen desmesuradamente, es decir si hay tantos niveles de malla como grados de división estrictamente necesarios. Ahora bien, los niveles de malla aparecen porque cada vez que se realiza un refinamiento local se crea un nuevo nivel de malla en la secuencia de mallas encajadas. Es decir, hasta ahora, dentro de los mallados estructurados, refinar implica añadir un nivel más a la secuencia de mallas.

Esta es la diferencia fundamental entre un algoritmo de refinamiento en mallas encajadas y uno de desrefinamiento. Al refinar se manejan sólo dos niveles de malla, el más fino de la secuencia de entrada y el que se está creando, con el cual se aumenta en una unidad el número de niveles. En un algoritmo de desrefinamiento es *toda* la secuencia de mallas la que se debe manejar porque la secuencia entera puede variar.

Con estos preliminares podemos preguntarnos: ¿No sería posible realizar el refinamiento local manejando *toda* la secuencia de mallas de forma que sólo se crease un nuevo nivel de malla si éste es *imprescindible*? Es decir, de forma análoga a como en el desrefinamiento se trabaja con toda la secuencia de mallas, ¿se podría refinar? ¿Bajo qué condiciones no se crearía un nuevo nivel de malla? ¿Es esto posible realizarlo con toda generalidad, para cualquier tipo de malla inicial?

No hemos resuelto todos estos interrogantes porque no es el objetivo de este trabajo y tampoco parece que tengan una única respuesta. Sin embargo, sí lo consideramos un trabajo interesante de cara al futuro. A continuación se apuntan dos posibles soluciones a este problema y se presenta alguna aplicación del mismo.

Llamamos a un tal refinamiento, *refinamiento estructurado* queriendo señalar de esa forma que es toda la estructura de mallas la que se maneja al refinar. No es, por tanto, el refinamiento en mallados estructurados del que se

ha estado hablando hasta ahora. En el refinamiento estructurado se introducirían los nodos en el nivel de malla *más bajo* donde esto sea posible.

En principio, se puede pensar desarrollar un algoritmo de refinamiento estructurado de tal forma que se modifique lo menos posible en algoritmo de refinamiento que ya tiene incorporado nuestro código de elementos finitos *Neptuno*. Ya se ha dicho que estamos utilizando una versión del algoritmo 4-T de Rivara para refinar. Este consta básicamente de las partes siguientes:

1.- Teniendo el cuenta el indicador de error por elemento que se ha calculado previamente y el parámetro de refinamiento se marcan las caras de los elementos (de la malla más fina) que serán divididos en 4 elementos hijos.

2.- Se asegura la conformidad de la malla que se está creando extendiendo la zona refinada.

3.- Se definen las conexiones nodales del nuevo nivel de malla.

Si se quiere construir un refinamiento estructurado tendiendo en cuenta el esquema anterior cabrían dos posibilidades:

1.- Pensar condiciones que permitan obtener una nueva secuencia de mallas *refinada* variando sólo el apartado 3.

2.- En caso de que lo anterior no sea posible, se puede pensar modificar el modo en que se logra la conformidad, que ahora afectaría a varios niveles de malla y también, obviamente, en cómo redefinir las conexiones nodales de los niveles afectados.

Respecto de la primera posibilidad apuntada, no está claro que se pueda aplicar a mallados triangulares de tipo general. Que se pueda aplicar o no un tal refinamiento, hasta donde sabemos depende de la geometría de malla inicial. Así si la malla inicial está compuesta de triángulos rectángulos isósceles, el lado mayor de cualquier triángulo generado por aplicación de los algoritmos 2-T ó 4-T de Rivara está predeterminado. En estas condiciones, además, el algoritmo 4-T de Rivara es equivalente a la aplicación doble del algoritmo 2-T. En este caso creemos que sí se puede desarrollar un algoritmo de refinamiento estructurado, cambiando únicamente la definición de un nuevo nivel de malla por la re-definición de las conexiones nodales de los niveles de malla afectados, y, si es necesario, definiendo el nuevo nivel de malla.

Este problema, es decir hallar condiciones que aseguren la equivalencia de los algoritmos 2-T y 4-T de Rivara, está intrínsecamente ligado al del refinamiento estructurado pues, en general, después de aplicar un desrefinamiento la secuencia de mallas encajadas que se obtiene no se puede lograr por el algoritmo 4-T de Rivara, sino por el 2-T y/o el 4-T combinados. Además, si lo que se pretende con un algoritmo de refinamiento de este tipo es variar la secuencia entera de mallas, se ha de poder cambiar el número de hijos de los elementos según convenga. Esto último sólo es posible si, en esa secuencia de mallas el algoritmo 4-T equivale al 2-T.

Demostramos a continuación algún resultado sobre condiciones bajo las cuales el algoritmo 4-T de Rivara coincide con la aplicación del 2-T dos veces consecutivas.

Propiedad.- Sea ***t*** un triángulo de ángulos α, β, y γ, con $\gamma \leq \beta \leq \alpha$. Si la aplicación en ese triángulo del algoritmo 4-T equivale a aplicar dos veces el algoritmo 2-T, entonces se verifica la relación:

$$\gamma \geq arcsen\left(\frac{sen\alpha}{2}\right)$$

Demostración:

Si en el triángulo ***t*** la aplicación de dos veces el algoritmo 2-T equivale al algoritmo 4-T, entonces $c \geq \frac{a}{2}$, siendo a el lado opuesto al ángulo α y c el opuesto al ángulo γ. En esas condiciones, por el teorema de los senos, se tiene $\operatorname{sen}\gamma \geq \frac{\operatorname{sen}\alpha}{2}$ de donde se tiene el resultado.■

Observación.- La condición anterior no es suficiente.

Demostración:

Sea ***t*** un triángulo de ángulos α, β, y γ, con $\gamma \leq \beta \leq \alpha$, como muestra la fig. 49. En ese triángulo 4-T coincide con dos veces 2-T si y sólo si

$$\left\{\begin{matrix}|AB|\geq|BM|\\|AB|\geq|AM|\end{matrix}\right\}$$

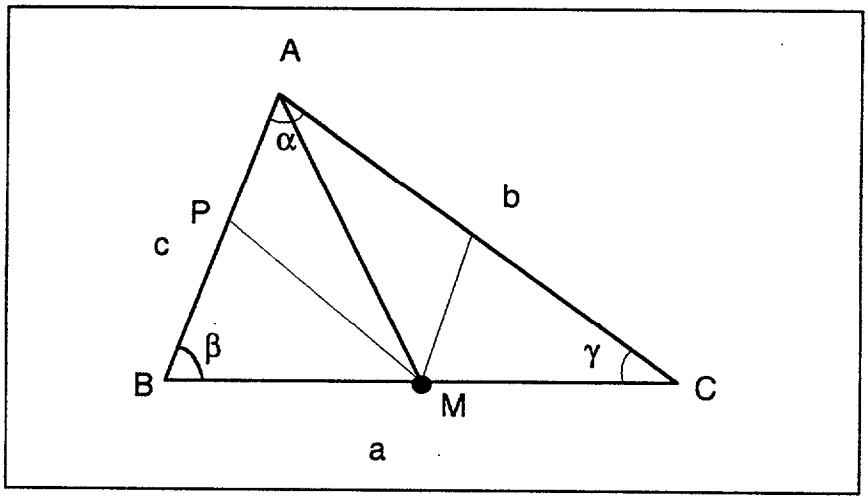


Figura 49.- Los algoritmos 4-T y 2-T.

La condición anterior sólo nos asegura la primera de las dos condiciones. Que ésta no es suficiente lo muestra la fig. 50. Suponiendo fijo el lado mayor BC, y considerando que γ es el ángulo menor, el dominio de variación para el vértice A sería el limitado por los segmentos MT, BM y el arco de circunferencia BST. La condición de la propiedad equivale a decir que el vértice A se encuentra en la región sombreada en la figura. Obsérvese, entonces que para A en la región limitada por el segmento RS y los arcos de circunferencia QR y QS, se cumpliría la condición

$$\gamma \geq arcsen\left(\frac{sen\alpha}{2}\right)$$

pero se tendría $|AB|<|AM|$, y, por tanto no equivaldría dos veces 2-T al 4-T.■

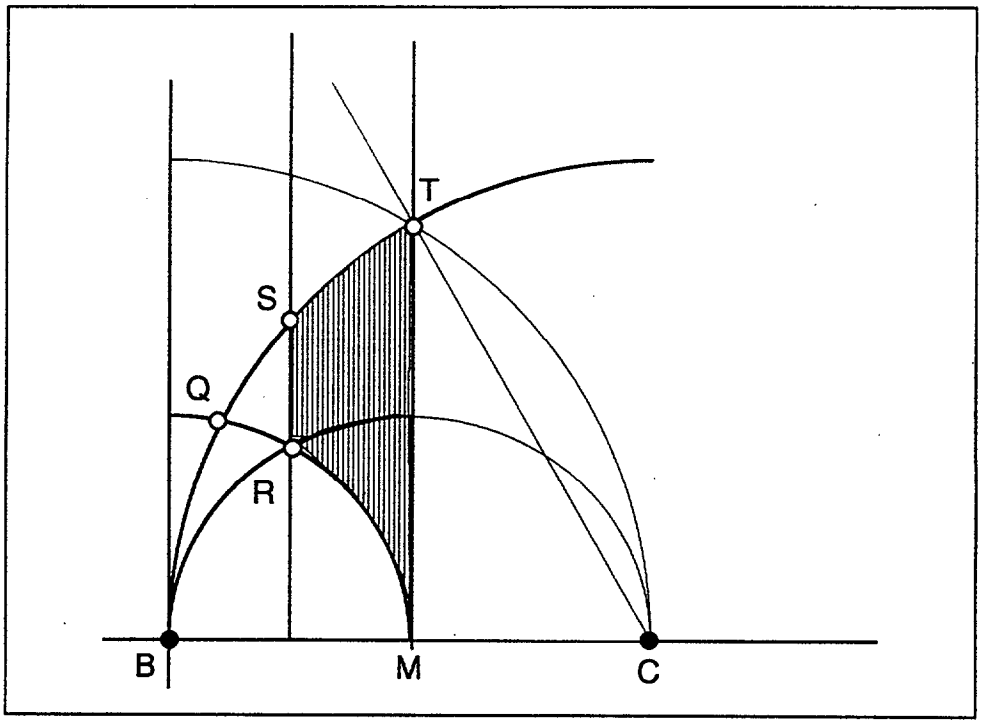


Figura 50.

Si la malla inicial se compone de triángulos isósceles rectángulos en cualquier malla obtenida por reiteración del algoritmo 2-T ó del 4-T se cumple la propiedad anterior, pues la división de un triángulo rectángulo isósceles produce siempre, como es obvio, triángulos rectángulos isósceles.

De todo lo anterior se deduce que la segunda posibilidad de *refinamiento estructurado* puede ser más general que la primera en el sentido de que se pueda aplicar a cualquier secuencia de triangulaciones anidadas.

En la fig. 51 se presenta un ejemplo de la aplicación del refinamiento estructurado del que hemos estado hablando. En el ejemplo se ha pretendido plasmar cómo funcionaría un algoritmo de refinamiento capaz de cambiar secuencias de triangulaciones. La primera línea de la figura muestra una secuencia de cuatro niveles de malla (la malla inicial se ha omitido) obtenida por el refinamiento 4-T estándar. Las siguientes significan las secuencias que se obtendrían por el refinamiento *estructurado* al refinar, respectivamente, la primera malla; la secuencia formada por las dos primeras mallas; y, en la última línea, la secuencia de la línea anterior.

Están resaltados los nodos propios de cada nivel. Véase en la figura cómo se introducen los nodos tan pronto como es posible. Se puede destacar que la malla más fina de las secuencias obtenidas mediante el refinamiento estructurado (últimas tres líneas) equivale al correspondiente

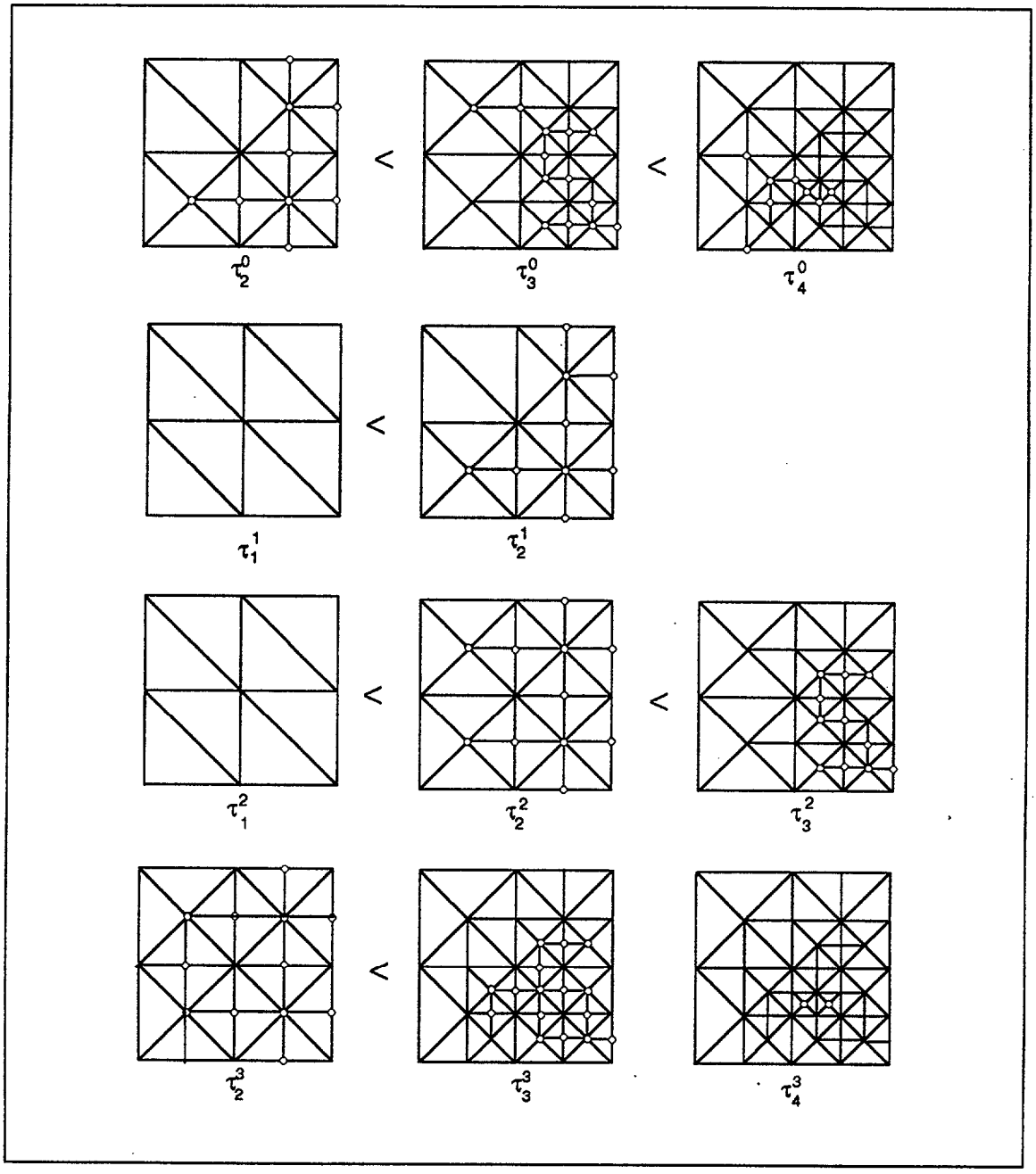


Figura 51.- Un ejemplo de refinamiento estructurado.

nivel de la primera línea (refinamiento estándar), con la única diferencia del número de nodos propios, que ahora es menor en los niveles más finos.

Es evidente que si se incorporase un refinamiento estructurado no sería necesario programar un algoritmo de compactación de mallas. Sin embargo, no lo hemos programado todavía. Aún así hay que señalar que utilizar una malla inicial de triángulos rectángulos isósceles tiene algunas ventajas siempre que el dominio del problema se ajuste a este tipo de elementos. Algunas de estas ventajas son:

1) Como la numeración local del lado mayor de todo triángulo creado por refinamiento de cualquier malla es conocida a priori, no sería necesario ni evaluar la longitud de los lados de un triángulo para saber quién es el lado mayor, ni almacenar esa numeración local. Recuérdese que esta numeración se guardaba en la matriz IXH, IXH(6,NE) = nº local del lado mayor del elemento NE.

2) Para cualquier secuencia de mallas se tiene que el ángulo mínimo permanece constante: $\alpha_{min} = 45º$. Esto nos proporciona una propiedad de alta regularidad en cualquier secuencia. La no degeneración de los triángulos queda asegurada.

3) Como se puede programar un algoritmo de refinamiento estructurado, el número de niveles de malla coincidiría con el mínimo grado de división necesario de la malla inicial, con el consiguiente ahorro de memoria.

4) De lo anterior es inmediato observar que el diámetro de cada nivel de malla dependería únicamente del diámetro de la malla inicial y del número de nivel particular. Como es conocido, si para cada triángulo t se define su diámetro h_t como la longitud de su lado mayor, el diámetro de una malla τ será $h_\tau = \min\{ h_t : t \in \tau \}$. De esta forma, si h_0 es el diámetro de la malla inicial τ_0, el diámetro de la malla obtenida tras n refinamientos sería:

$$\frac{h_0}{2^n} \leq h_n \leq \frac{h_0}{2^n}\sqrt{2}$$

Esto último permitiría el cálculo automático del diámetro de malla, detalle particularmente útil en problemas evolutivos, en los que, por cuestiones de estabilidad, el paso de tiempo admisible depende del diámetro de malla.

Sin embargo, es más útil que el refinamiento no dependa de la geometría de la malla inicial. En este sentido para programar un refinamiento estructurado habría que asegurar la conformidad de todos los niveles de malla intermedios afectados, de forma semejante a como se logra en la actualidad la conformidad del naciente nivel de malla con el refinamiento estándar. Un esquema de cómo actuaría ese algoritmo podría ser el siguiente:

ENTRADA: Secuencia $T = \{ \tau_1 \leq \tau_2 \leq ... \leq \tau_n \}$

① Dependiendo de los indicadores de error y del parámetro de refinamiento se marcan las caras de los elementos que han de ser refinados.

② Bucle en niveles de malla, desde el nivel más alto (n2) afectado hasta el más bajo (n1). Sea el nivel de malla j: Desde j = n2 hasta n1, hace:

Se asegura la conformidad de la nueva subsecuencia

$T^j = \{ \tau_1 \leq \tau^j_2 \leq ... \leq \tau^j_j \}$ extendiendo la zona que se refina.

Fin del bucle en niveles de malla.

③ Bucle en niveles de malla, desde el nivel más bajo (n1) afectado hasta el más alto (n2). Sea el nivel de malla j: Desde j = n1 hasta n2, hace:

Se definen nuevas conexiones nodales para el nivel τ^j_j

Se heredan los cambios a la subsecuencia final. Se obtiene una nueva secuencia: $T^j = \{ \tau_1 \leq ... \leq \tau^{n1}_{n1} \leq ... \leq \tau^{j-1}_{j-1} \leq \tau^j_j \leq ... \leq \tau^j_n \}$.

Fin del bucle en niveles de malla.

SALIDA: Secuencia refinada $T' = \{ \tau_1 \leq \tau'_2 \leq ... \leq \tau'_m \} =$

$= T^j = \{ \tau_1 \leq ... \leq \tau^{n1}_{n1} \leq ... \leq \tau^{n2}_{n2} \}$.

Se puede señalar que la condición necesaria y suficiente para que se cree un nuevo nivel de malla es que alguna cara propia de la malla más fina

de la secuencia de entrada resulte marcada en los pasos ① ó ② del algoritmo. El esquema anterior es, quizá, excesivamente sencillo y se echan en falta muchos detalles por concretar. Se puede notar que los niveles de malla $n1$, y también el $n2$, pueden variar en el proceso y, en principio, hay que asegurar en el paso ② la conformidad de toda una subsecuencia de niveles de malla.

a2

La eficiencia algorítmica

El objetivo principal de este apéndice es resumir algunos conceptos básicos sobre la eficiencia algorítmica.

Medir la *eficiencia* de un algoritmo es medir la cantidad de recursos necesarios para su ejecución, a efectos de cotejarla con la de otros algoritmos ya inventados o por inventar.

Suelen elegirse como medida de eficiencia factores que se pueden definir formalmente en términos de programación y que tienen la máxima influencia en el coste real de la ejecución: fundamentalmente el tiempo de cálculo y, en menor medida, la cantidad de memoria interna. La cantidad de memoria secundaria puede ser relevante en ciertas ocasiones. En programación paralela se considera también a veces como recurso a medir el número de procesadores.

El tiempo de ejecución de un programa se puede conocer para ciertos datos simplemente ejecutando el programa, pero la información que se obtiene es muy escasa: no se sabe, entonces, qué puede ocurrir en otra máquina, con otro compilador, y, sobre todo, con otros datos. Todos esos factores influyen en la eficiencia de un algoritmo. De todas formas la variable fundamental es el tamaño de los datos, y en función de este tamaño es como se suele medir la eficiencia de los algoritmos, y se logra que este análisis sea independiente de otros factores haciéndolo independiente de factores multiplicativos, es decir, del producto por constantes. Este grado de abstracción se consigue clasificando las expresiones que denotan el uso de recursos en función de su velocidad de crecimiento. Se acostumbra a denominar $T(n)$ al tiempo de ejecución de un programa con una entrada de tamaño n.

Al expresar la eficiencia de un algoritmo en función del tamaño de los datos hay que decidir entre si nos estamos refiriendo al caso peor, al caso mejor, o, en algún sentido, a un caso medio. El usado comúnmente es el análisis del caso peor.

En cuanto al análisis propiamente dicho de la cantidad de recursos requerida por un algoritmo, éste se realiza en base a las instrucciones que lo componen. Algunas ideas son las siguientes:

La evaluación de una expresión requiere como tiempo la suma de los tiempos de ejecución de las operaciones, o llamadas a funciones, que la componen.

Una asignación a una variable no estructurada, o una operación de escritura de una expresión, suelen requerir el tiempo de evaluar la expresión más unas operaciones adicionales de gestión interna de coste constante.

Una lectura de una variable no estructurada suele requerir tiempo constante.

El coste de una secuencia de instrucciones es naturalmente la suma de los costes de las instrucciones, que suele coincidir con el máximo de los costes.

Para un bucle es necesario sumar los costes de cada vuelta, sin olvidar que cada una requiere una evaluación de la expresión indicada en la condición del bucle.

Una llamada a un procedimiento o función requiere el análisis previo del mismo; si es recurrente, puede aparecer una ecuación recurrente, de las que no se hablarán en este apéndice.

Las notaciones asintóticas y sus propiedades

En este apartado se recuerdan algunas de las notaciones más corrientes para clasificar funciones en base a su velocidad de crecimiento, también llamada orden de magnitud. Se basan en el comportamiento en casos límite y definen el llamado *coste asintótico de los algoritmos.*

Sólo se incluyen aquí algunas de las más usadas. Otras notaciones asintóticas se pueden hallar, por ejemplo, en [Balca92], [AhHoU83]. Todas las funciones que aparezcan serán funciones de N^+ en N^+, si bien estas

definiciones son válidas para funciones de N^+ en R, siempre que tomen valores superiores a la unidad.

La O mayúscula

La notación *O* mayúscula, *O(f)*, denota el conjunto de funciones g, que "crecen a lo más tan rápido como" f, es decir, la funciones g tales que, módulo constantes multiplicativas, f llega a ser a partir de algún momento una cota superior para g. Su definición formal es:

$$O(f)=\left\{g \,\middle|\, \exists c_0 \in \mathrm{R}^+, \exists n_0 \in N \neg \forall n \geq n_0, g(n) \leq c_0 \cdot f(n)\right\}$$

Las propiedades de la notación *O(f)*, para cualesquiera funciones f, g y h son las siguientes:

1.- *Invarianza multiplicativa:* $\forall c \in R^+, \quad g \in O(f) \Leftrightarrow c \cdot g \in O(f)$.

2.- *Reflexividad:* $f \in O(f)$.

3.- *Transitividad:* $\left.\begin{array}{l} h \in O(g) \\ g \in O(f) \end{array}\right\} \Rightarrow h \in O(f)$.

4.- *Criterio de caracterización:* $g \in O(f) \Leftrightarrow O(g) \subseteq O(f)$.

5.- *Regla de la suma para O:* $O(f+g)=O(max(f,g))$.

6.- *Regla del producto para O:* $\left.\begin{array}{l} g_1 \in O(f_1) \\ g_2 \in O(f_2) \end{array}\right\} \Rightarrow g_1 \cdot g_2 \in O(f_1 \cdot f_2)$.

7.- *Invarianza aditiva:* $\forall c \in R^+, \quad g \in O(f) \Leftrightarrow c+g \in O(f)$.

Como se ve, $g \in O(f)$ indica que el crecimiento de g no supera al de f.

La o minúscula

A diferencia de la notación anterior, $g \in o(f)$ indica que el crecimiento de g es estrictamente más lento que el de f. Su definición formal es:

$$o(f) = \left\{ g \,\middle|\, \forall c_0 \in \mathrm{R}^+, \exists n_0 \in N \neg \forall n \geq n_0, g(n) \leq c_0 \cdot f(n) \right\}$$

Es claro que el rango de constantes multiplicativas c_0 *interesantes* en la definición está formado por constantes inferiores a la unidad.

La *o minúscula* cumple las propiedades de invarianza multiplicativa, invarianza aditiva, antirreflexividad, transitividad, y, además, las tres siguientes:

i) $g \in o(f) \Leftrightarrow g \in O(f) \ \wedge \ f \notin O(g)$

ii) $g \in o(f) \Leftrightarrow O(g) \subseteq o(f)$

iii) Caracterización por límites:

$$g \in o(f) \Leftrightarrow \lim_{n \to \infty} \frac{g(n)}{f(n)} = 0$$

Otras notaciones asintóticas son:

Las notaciones Ω

Para denotar cotas inferiores de forma análoga a las cotas superiores, se dispone de las notaciones Ω. Existen dos distintas. Denotaremos a la primera $\Omega_K(f)$ como hace [Balca92] en honor de D. Knuth, quien la propuso. Sus definiciones son:

$$\Omega_K(f) = \left\{ g \,\middle|\, \forall c_0 \in \mathrm{R}^+, \exists n_0 \in N \neg \forall n \geq n_0, g(n) \geq c_0 \cdot f(n) \right\}$$

$$\Omega_{\infty}(f)=\left\{g \,\middle|\, \left(\forall c_0 \in \mathrm{R}^{+}\right) \wedge \left(\forall n_0 \in N\right),\ \exists n \geq n_0, \neg\, g(n) \geq c_0 \cdot f(n)\right\}$$

Las notaciones Θ

Denotan las funciones con la misma tasa de crecimiento que *f*:

$$\Theta(f)=O(f) \cap \Omega_K(f)$$

La clase $\Theta(f)$ es un subconjunto de $O(f)$ y se denomina a veces *orden de magnitud de f.*

Muy frecuentemente resulta difícil evaluar con exactitud el orden de magnitud o coste de un algoritmo, $\Theta(f)$, y nos hemos de conformar tan sólo con una cota superior de él: $O(f)$.

Notaciones asintóticas con varios parámetros

Al analizar un algoritmo, puede suceder que su tiempo de ejecución dependa simultáneamente de más de un parámetro. Las definiciones de las notaciones asintóticas anteriores se pueden generalizar a funciones de varias variables. Supongamos que las funciones que aparecen en este apartado son funciones de $\mathrm{N}^{+} \times \mathrm{N}^{+}$ en N^{+}. Entonces se define:

$$O\left(f(m,n)\right)=\left\{g \,\middle|\, \exists c_0 \in \mathrm{R}^{+}, \exists\, m_0, n_0 \in N \neg \left(\forall n \geq n_0\right) \wedge \left(\forall m \geq m_0\right), g(m,n) \leq c_0 \cdot f(m,n)\right\}$$

El resto de las notaciones asintóticas para funciones de varios parámetros se hace de forma similar a la definición anterior.

Formas de crecimiento frecuentes

La escala con arreglo a la cual se clasifica el uso de recursos de los algoritmos consiste en una serie de funciones elegidas entre las que se pueden definir mediante operaciones aritméticas, logaritmos y exponenciales.

Las formas más importantes de crecimiento asintótico son las siguientes:

Logarítmico: $\Theta(\log n)$

Lineal: $\Theta(n)$

Quasilineal: $\Theta(n \cdot \log n)$, aunque a veces también se entiende por quasilineal a $\Theta\left(n \cdot (\log n)^k\right)$ con k fijo.

Cuadrático: $\Theta\left(n^2\right)$

Cúbico: $\Theta\left(n^3\right)$

Polinómico: $\Theta\left(n^k\right)$ con k fijo.

Exponencial: $\Theta\left(k^n\right)$ con k fijo.

Otras formas de crecimiento que pueden aparecen con cierta frecuencia incluyen $\Theta\left(\sqrt{n}\right)$, $\Theta(n!)$, o bien $\Theta\left(n^n\right)$.

A falta de posibilidades de comparación, suele admitirse como algoritmo eficiente el que alcanza un coste quasilineal. Cualquier coste superior al polinómico, e incluso un coste polinómico con un grado elevado, es un coste que se considera intratable; en efecto, incluso sobre problemas de tamaño moderado, los algoritmos que exhiben ese coste suelen exigir ya unos recursos prohibitivos.

a3

Publicaciones más importantes

A-12

ALGORITHMS, SOFTWARE, ARCHITECTURE

Information Processing 92

Proceedings of the IFIP 12th World Computer Congress
Madrid, Spain, 7-11 September 1992

VOLUME I

Edited by

JAN VAN LEEUWEN
Department of Computer Science
University of Utrecht
Utrecht, The Netherlands

1992

NORTH-HOLLAND
AMSTERDAM • LONDON • NEW YORK • TOKYO

Algorithms, Software, Architecture / J. van Leeuwen (Editor)
Information Processing 92, Volume I
Elsevier Science Publishers B.V. (North-Holland)

Derefinement algorithms of nested meshes

A. Plaza[a], L. Ferragut[b] and R. Montenegro[a]

[a]Department of Mathematics, University of Las Palmas de Gran Canaria, Campus Universitario de Tafira, 35017-Las Palmas de Gran Canaria, Spain

[b]Department of Applied Mathematics and Informatic Methods, Politechnic University of Madrid, c / Rios Rosas 21, Madrid, Spain

Abstract

In this paper a new derefinement algorithm of nested triangular meshes is presented and discussed. This algorithm is combined with a local refinement. The refinement / derefinement combination, that we call *readaptive process,* is very useful to solve time-dependent problems in which moving refinement areas are required. The fact of using nested meshes enables us to use the multigrid method in order to solve the equation system associated to the finite element method. Moreover, a readaptive process can be used to obtain the best piece-wise triangular support to interpolate a given two dimensional function.

Keyword Codes: G.1.8; G.4; I.1.2
Keywords: Partial Differential Equations; Mathematical Software; Algorithms

1. INTRODUCTION

In the last years, the efficiency of the adaptive finite element method has been proved, particularly to solve problems in which the solution presents a singularity [1-4]. However, when studying a time-dependent problem, if this dependency implies a change of the top gradient area, the existence of a large number of points creates a serious difficulty. Many of these nodes -though necessary in any past time- are *useless* in the present moment. This fact is very important because an increment in the number of nodes in the mesh implies an increase in the number of equations of the associated algebraic system to be solved, and in the necessary memory space. If we think that, using element by element (EBE) methods, more than 90% of CPU time is employed in solving this equation system, it seems necessary to study a derefinement algorithm able to remove *dupe* nodes and to be combined with a local refinement in this kind of problems.

On the other hand, the multigrid method presents a very convenient feature in number of operations required for solving the equation system [5,6]. Hence, the suitability of using nested meshes both at the refining and at the derefining stages. A new derefinement algorithm of nested meshes able to be combined with a refinement algorithm and be implemented with the multigrid method is presented in this work. Now, the meshes generated by a readaptive process are more flexible than those obtained by local refinement only, and the number of nodes remains bounded during the whole process.

Logically, for each refinement algorithm, a parallel derefinement algorithm can be thought, if the structure of nested meshes is to be maintained. We want point out here that for every refinement algorithm not always an inverse derefinement algorithm can be found. We have used a version of the 4-T refinement algorithm of M.C. Rivara [7,8], that is implemented in our code Neptuno [1].

2. PREVIOUS DEFINITIONS. PROPERTIES.

Let $T = \{ \tau_1 < \tau_2 < ... < \tau_n \}$ be a sequence of nested meshes and τ_j any triangulation of T. One node N of τ_j will be called a *proper node* of τ_j if it does not belong to any previous mesh, i.e. $N \in \tau_j$ and $N \notin \tau_i$, $\forall i < j$. In other case, N will be called *inherited node* in τ_j.

Similarly, any edge or element is *proper* or *inherited* in each mesh. Thus, if one edge is divided in two, in the refinement process, we shall say that the former edge is the *father edge* and the latter are the *son edges*. *Father elements* and *son elements* are defined in the same manner. Moreover, if an edge is proper in a mesh, said τ_j, we shall distinguish between *external* and *internal* edges: a proper edge is external if it has a father edge, and it is internal if it has not. In figure 1 an example of these definitions is presented. Nodes N_1 and N_2 are proper nodes in τ_j; edges f_1 and f_2 are external in τ_j, f_3 is internal and c_3 is inherited; finally, element e_1 has three sons and e_2 has two.

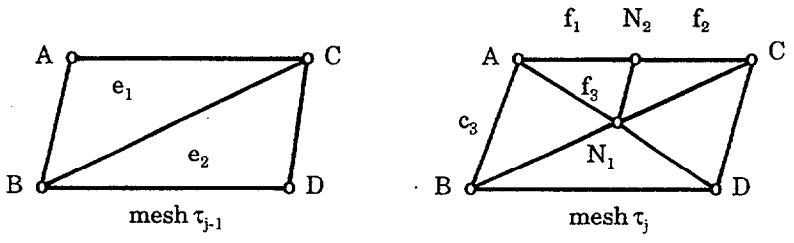


Figure 1. Example of refinement.

It is easy to observe that:

i) any proper element of some triangulations either it is inherited in the following triangulations or it has its sons there.

ii) if an element has not sons, it belongs -as proper or as inherited- to the finest mesh of the irregular nested triangulations.

These definitions and properties are important because in contrast to the refinement algorithm in which only two levels of meshes are involved -the last mesh created and the new one that is being created at that moment- in the derefinement algorithm we obtain, in general, a new sequence of irregular nested triangulations: all level of meshes are involved in this process. For this reason, the *family* or *genealogy* of each element must be considered.

The fundamental property of the derefinement algorithm is the following:

iii) *Only those elements without successors, that is those which belong to the current finest mesh, can be eliminated.*

3. DATA STRUCTURE

The structure of nested meshes is defined in the code Neptuno by three vectors, that will be called *intermediate vectors:* IMNODE, IMFACE and IMELEM. In these vectors the numbers of nodes, edges and elements of each level of mesh are kept. It is sufficient to keep the proper nodes of each level because if one node belongs to a particular mesh, it belongs to the following meshes as well. This is false with respect to edges and elements. Therefore, an edge or an element appears as many times as levels of meshes it belongs to.

The family of each edge is given by the matrix IR[1:3,1:NUMF] where NUMF is the total number of edges in the structure of meshes. For each edge, IR reports the numbers of the son edges and of the father edge.

The family of each element is defined by the matrix IXH[1:6,1:NUMEL] being NUMEL the global number of elements in our structure of meshes. For each element, IXH gives us its sons, the local number of the longest side and, finally, the number of its father.

Furthermore, we need three vectors that will be called *derefinement indicators* or *level vectors* and that will find out when a node, edge or element has to be cancelled out, and at the same time, the level of the mesh in which it is proper. We point out here that this level is well defined. These vectors are: NODES, for the nodes, NFACES, for the faces and NELES for the elements.

The numbers of nodes, edges and elements eliminated by derefine procedure will be kept in other three vectors that will be called *sack vectors:* NNSAC, NFSAC and NESAC. These numbers will be used on next refinements.

Finally, and in order to facilitate the computation of the derefinement condition, we introduce a new vector, said IEX[1:NUMP] where NUMP is the number of nodes in the mesh. For each node IEX reports the *surounding edge*, that is the number of the edge in which that node is in the middle point.

We should stress that the size of the vectors and matrices used by Neptuno is

variable. When refining, the size grows; whereas at derefining the size decreases. Moreover the space of memory used can be optimized, e. g. the intermediate vectors are not neccesary, but they are very convenient for ease and efficiency of the multigrid method implementation.

4. THE DEREFINEMENT ALGORITHM

In general, we have a sequence of nested meshes $T = \{ \tau_1 < \tau_2 < ... < \tau_n \}$ and we want to obtain another sequence after derefining T, $T' = \{ \tau_1 < \tau_2' < ... < \tau_m' \}$, i.e.,

i) $m =< n$

ii) $\forall\ \tau_k' \in T', \exists\ \tau_i,\ \tau_j \in T$, such as $\tau_i < \tau_k' < \tau_j$

In this case, we will say that the secuence T' is coarser than T, or T finer than T', and write $T' < T$. This relation is a partial order relation in the family of sequences of triangular meshes over a given bidimensional domain.

Schematically, the derefinement algorithm can be described as follows:

INPUT: Secuence $T = \{ \tau_1 < \tau_2 < ... < \tau_n \}$

For j=N to 2, do:

1. For each proper node of mesh τ_j, the derefinement condition is evaluated and the nodes and edges able to be eliminated are pointing out.
2. Conformity of the new level *j* is assured, minimizing the derefined area.
3. New nodal conections, and new families of edges and elements are defined for the level τ_{j-1} and for the derefined level of τ_j, said τ_j^*.
4. The changes in the current level *j* are inherited to the following ones.
5. It is obtained a new sequence of nested meshes, T_j, that is the new input in the next iteration of the loop on meshes.

OUTPUT: Sequence $T' = \{ \tau_1 < \tau_2' < ... < \tau_m' \}$

5. DISCUSSION OF THE ALGORITHM

A new structure of meshes coarser than the preceding one, is obtained in each iteration of the loop. Thus, the number of different sequences that appear along the derefine procedure is the number of level of meshes minus one. This can be expresed as follows: $T > T_n > \ldots > T_2 = T'$. So, the mesh τ_j^* belongs, in general to an intermediate sequence; only at the last iteration of the loop (j=2) τ_2^* is in the derefined sequence T', and then τ_2^* is equal to τ_2'.

It is worth to be noted that only proper nodes are elegible in each mesh-level, and out of these, only those suitable to be cancelled out are taken for evaluation. This means that if one node cannot be cancelled this implies that any neighbouring nodes cannot be cancelled either. In this manner the nestedness of the new sequence is assured. An example of this question can be observed in figure 1: if N_1 has to stay, nodes A,B,C and D will also stay. This allows us to

evaluate the derefinement condiction in a number of nodes minimal, and only once for each of those.

Regarding the derefinement condition, the following one has been used: the absolute difference between the numerical solution at node N and the interpolated solution of the ends of its surounding edge, is checked. If this absolute difference is less than a constant -that can be fixed for each program run-, the node N can be cancelled. This imposed constant will be called epsilon. However, the relative difference can also be used.

The conformity of all meshes in each sequence is assured to maintaining some nodes that otherwise, and concerning the derefinement condition, could have been cancelled. In fact, if a node N belongs to the longest edge of an element in which in other edge there is another node, the node N remains. For instance, in figure 1, if N_2 remains, N_1 remains too.

In the derefinement process some new elements may arise; these elements did not exist in the input sequence. For instance, in figure 1, if node N_2 is cancelled, the element AN_1C was not in τ_j . These elements will be called half-sons. At derefining, an element that had four elements might turn out to have three, two, or none sons. The definition of the new nodal connections is obtained considering all of different possibilities and their old nodal connections.

The algorithm cheks whether all proper nodes of every mesh are going to be eliminated. In this case, it is not necessary to define new nodal connections, but to take the intermediate vectors in order to compress them. That mesh level is then eliminated. For this reason, the number of meshes in the output structure can be made lesser than in the input sequence.

At the end of each iteration, the modifications of the new nodal connections have to be passed on to the following levels of meshes. This is obtained by changing the intermediate vectors. Therefore it is necessary to manage the derefinement indicators. A new intermediate sequence is obtained and taken out as input in the next iteration.

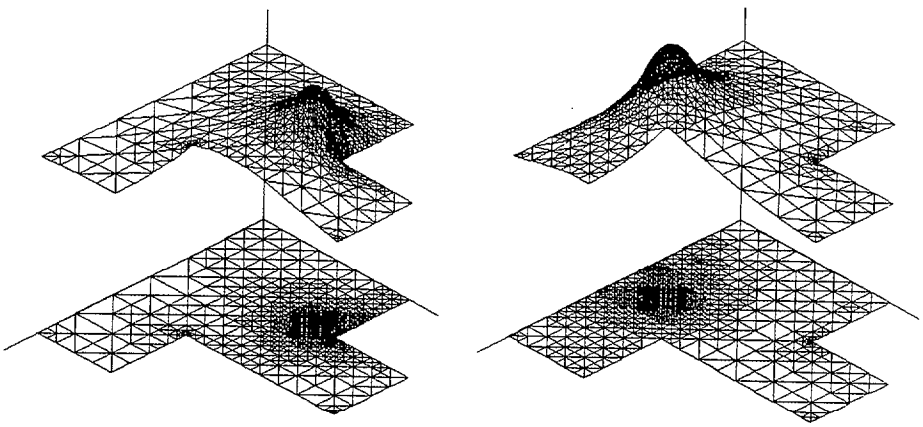
6. NUMERICAL RESULTS

Some numerical examples are presented in this section.

Figure 2 shows some meshes and the solution in different time of events of a Poisson problem in which the function of the second member is time-dependet. The domain geometry is non symetrical and we have Dirichlet and Neumann conditions in the boundary. Maximun number of nodes managed was 3397, and maximun number of mesh-level was 24. The maximum number of nodes eliminated by one application of the derefinement algorithm was 2748 and the minumun 252. Parameter epsilon used was 0.0009 with relative difference as derefinement condition.

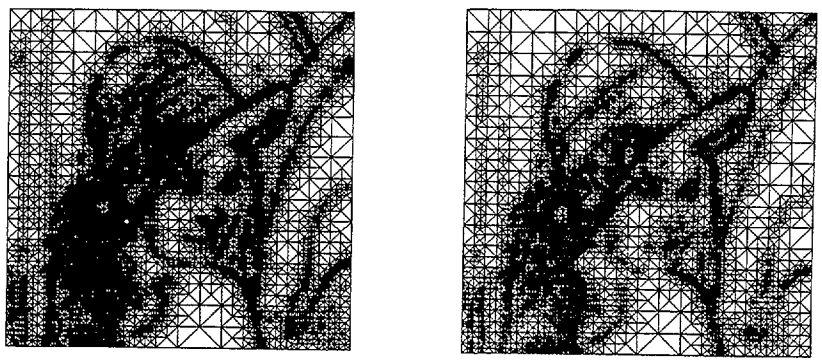
Figure 3 shows the efficiency of the derefinement algorithm to obtain the best piece-wise support to interpolate a given two-dimensional function. The function

selected here is a classical image of 256x256 pixels, see for example [9]. A sequence of eight nested meshes has been generated automatically. In each node we allocate the corresponding grey level value, from 0 to 255. After, the derefinement algorithm has been applicated using different values for epsilon and absolute difference as derefinement condition.



(a) (b)

Figure 2. Derefined meshes and solution of a quasievolutive Poisson problem. (a) t=0 s, 674 nodes on the mesh, (b) t=6 s, 823 nodes.



(a) (b)

Figure 3. Different derefined meshes to approach a black-and-white image. (a) ε=15; 13892 nodes; (b) ε=25; 9112 nodes.

7. CONCLUSIONS

The refinement/derefinement combination is very useful to solve time-dependent problems in which moving refinement areas are required. The structured derefinement algorithm enables us to use the multigrid method in order to solve the equation system associated to the finite-element method. The aditional computation time required by this algorithm amounts less than 1% of total execution time. Only a scalar optimization at compilation time has been done on a Stardent 3000 computer.

As another application, this derefinement algorithm can be used for local refining. Derefining after global refinement is thereby equivalent to a local refinement procedure, not requiring the use of error indicators. This implies clear advantages in the class of problems for which error indicators cannot be found or their computation in too costly.

Finally, this algorithm can also be used in order to obtain the best piece-wise support to interpolate a given two-dimensional function, or to approach the initial solution in an evolutive problem.

8. REFERENCES

1 L. Ferragut, A solution to the programming problem of self-adaptive finite element methods (in spanish), Anales de Ing. Mecánica, Year 5, n. 1, pp. 201-206 (1987).

2 J.P. de S.R. Gago, D.W. Kelly, O.C. Zienkiewicz and I. Babuska, A Posteriori Error Analisys and Adaptive Processes in the Finite Element Method; Part II: Adaptive Mesh Refinement, Int. Jour. Num. Meth. in Eng., Vol. 19, pp. 1621- 1656 (1983).

3 D.W. Kelly, J.P. de S.R. Gago, O.C. Zienkiewicz and I. Babuska, A Posteriori Error Analisys and Adaptive Processes in the Finite Element Method; Part I: Error Analisys, Int. Jour. Num. Meth. in Eng., Vol. 19, pp. 1593-1619 (1983).

4 J.Z. Zhu and O.C. Zienkiewicz, Adaptive Techniques in the Finite Element Method, Comp. in App. Num. Meth., Vol. 4, pp. 197-204 (1988).

5 W. Hackbush and V. Trottengurg (eds.), Multigrid Methods, Lectures Notes in Mathematics, Springer-Verlag, Berlin, 1982.

6 W. Hackbush and V. Trottengurg (eds.), Multigrid Methods II, Lectures Notes in Mathematics, Springer-Verlag, Berlin, 1982.

7 M.C. Rivara, A Grid Generator based on 4-triangles conforming mesh refinement algorithms, Int. Jour. Num. Meth. in Eng., Vol. 24, 1343-1354 (1987).

8 M.C. Rivara, Selective refinement / derefinement algorithms for sequences of nested triangulations, Int. Jour. Num. in Eng., Vol. 28, pp. 2889-2906 (1987).

9 R. Aravind and Allen Gersho, Image Compression Based on Vector Quantization with Finite Memory, Opt. Eng. Vol. 26, No. 7, pp. 570-580, July (1987).

Numerical Methods in Engineering '92

Proceedings of the First European Conference on
Numerical Methods in Engineering
7-11 September 1992, Brussels, Belgium

Edited by

Ch. Hirsch
Vrije Universiteit Brussel
Brussels, Belgium

O.C. Zienkiewicz
University College of Swansea
Swansea, UK

E. Oñate
Polytechnical University of Catalonia
Barcelona, Spain

1992

ELSEVIER
AMSTERDAM • LONDON • NEW YORK • TOKYO

Numerical Methods in Engineering '92
Ch. Hirsch et al. (Editors)

AN ADAPTIVE REFINEMENT/DEREFINEMENT ALGORITHM OF STRUCTURED GRIDS FOR SOLVING TIME-DEPENDENT PROBLEMS

A. Plaza[1], R. Montenegro[1] and L. Ferragut[2]

[1]Department of Mathematics, University of Las Palmas de Gran Canaria, Campus Universitario de Tafira, 35017-Las Palmas de Gran Canaria, Spain
[2]Department of Applied Mathematics and Informatic Methods, Polytechnic University of Madrid, c / Rios Rosas 21, Madrid, Spain

The adaptive refinement/derefinement processes of structured grids are very useful to solve time-dependent problems in which moving refinement areas are required. In this paper some definitions and properties are presented in order to explain our derefinement algorithm. The efficiency of this algorithm is showed in some numerical examples: a quasi-evolutive test problem and an evolutive convection-diffusion problem with dominant convection. In this last case, we present the principal numerical aspects about the implicit formulation that has been used.

1. INTRODUCTION

The traditional approach to the solution of time-dependent problems using adaptive finite element methods has been revealed very useful to solve this kind of problems (ref. 1-3).

However, if only a local refinement is used to approach the solution at each time event, if the refined areas change with the time, the appearance of a large number of nodes creates a serious difficulty. Many of these nodes -though necessary in any past time- are useless in the present moment.

As is well known, the number of unknowns of the associated algebraic system to the finite element method is about the number of nodes of the mesh multiply by the number of degrees of freedom/node of our problem. So, an increment in the number of nodes in the mesh implies an increase in the number of equations of the system to be solved. Thereby in the topics of structured meshes it seems necessary to develop a derefinement algorithm able to remove dupe nodes, to get a good approximation of the numerical solution obtained in previous time step and to be combined with a local refinement. We have used triangular elements with three nodes and a version of the 4-T algorithm of Rivara (ref. 4-5) at the refining. The derefinement algorithm can be understood as the inverse algorithm of the refinement one.

With this combination (refinement and derefinement) that we call readaptive process (ref. 6), we get families of sequences of structured meshes more flexible than those obtained by local refinement only and with the advantage that the number of equations does not increase so much during the whole evolutive process.

Moreover, the fact of using nested grids enables us to use the multigrid method in order to solve the equation system associated to the finite element method (ref. 7-8).

In the paper there are four main sections as follows. In the following, a readaptive process is described in a general manner. Some definitions, relevant properties and the data structure used are followed by the description of the derefinement algorithm. This algorithm has been presented by the authors in (ref. 6).

The second section concerns to the convection-diffusion problem. A previous numerical study is necessary: the implicit formulation used is shown, followed by a discussion of the stability and consistency as it can be founded in (ref. 9). Afterwards, the error indicators used and the adaptive strategy employed are explained.

Some numerical results are given in the third section. We present two examples as application of the readaptive algorithm. In the first, the numerical solution for a Poisson's test-problem with a time-dependent function on the second member is presented. After, the solution of an evolutive convection-diffusion problem with dominant convection is given. In this case, we prove too the efficiency of the refinement/derefinement algorithm to approach the initial solution of the problem.

In the last section some remarkable conclusions are emphasized.

2. THE READAPTIVE PROCESS

2.1 General Aspects

In this section, our purpose is to explain what is a readaptive process. Fig. 1 summarizes a such procedure:

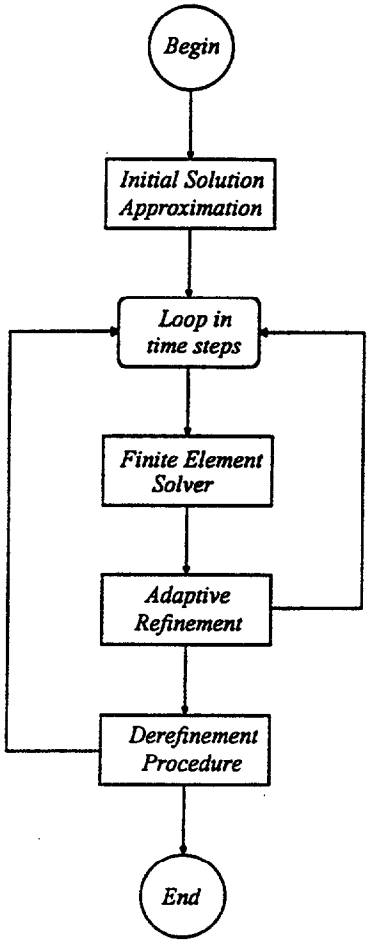


Figure 1: Principal modules of a readaptive process in a time-dependent problem.

In order to obtain the best piece-wise support to approximate the *initial solution*, some global refinements to achieve the function are performed, after a derefinement application optimize the mesh. This procedure is very convenient for initial solutions with high gradient areas. An initial mesh with a minimum number of nodes is attained by this way. Besides, only it is necessary to introduce an input data capable to define the geometry of the domain.

The number of *time steps* of each program run is fixed by the user at the beginning. This number should depend on each problem. In evolutive problems it can be set to a value of several thousands.

For each time step a *finite element solver* is applied to obtain the numerical solution for the current grid. Here, if the problem is evolutive, an automatic calculus of the time increment is used for verifying the stability condition of the formulation for the current grid. In the case that the problem is quasi-evolutive, the time increment can be fixed at the beginning. Finally, in this module we solve the equation system using a multigrid method.

When we have the numerical solution an *adaptive refinement* is made using an error indicator depending of the problem. Several adaptive refinements in the same time event can be thought. However, as usually the time increment is very small, the changes in numerical solution are very small too. For this reason, after each adaptive refinement we continue with the *loop in time steps*. This process is executed few times depending on the adaptive strategy that we have chosen. The number of local refinement can be fixed by the user or can be change automatically in function of the permitted maximum number of nodes.

Afterwards, in order to manage a reasonable number of nodes, a *derefinement procedure* must be used before the next loop in time steps. This module includes a geometrical problem: the derefinement algorithm, and an algebraic problem: the renumeration of the equations associated to the new sequence of structured meshes.

2.2 Definitions, Properties and Data Structure

Let $T = \{ \tau_1 < \tau_2 < \ldots < \tau_n \}$ be a sequence of structured triangular grids and τ_j any triangulation of T. One node N of τ_j will be called a *proper node* of τ_j if it does not belong to any previous mesh, i.e. $N \in \tau_j$ and $N \notin \tau_i$, $\forall\, i < j$. In other case, N will be called *inherited node* in τ_j. Similarly, edges and elements are named in each level of mesh.

If an edge is divided in two at refining, it is called *father edge* of these, and these are the *son edges* of the former. *Father elements* and *son elements* are defined in the same manner. As we are using the 4-T algorithm of Rivara at refining, an element has four sons or less.

When an element is refined, inside it some edges appear. These edges are called *internal edges*. In the boundary of the element some edges appear too. Now these edges are called *external edges*. Fig. 2 shows an example of refinement. There, nodes N_1, N_2 and N_3 are proper ones in τ_j; edges f_1 and f_2 are external in τ_j, f_3 is internal and c_1 is inherited.

Some properties in the sequences of structured

triangular grids are:
i) any proper element of some triangulation either it is inherited in the following triangulation or it has its sons there.
ii) if an element has not sons, it belongs to the finest mesh of the irregular nested triangulations.
iii) only those elements without successors can be eliminated. In other words: only elements that belongs to the finest mesh can be eliminated.

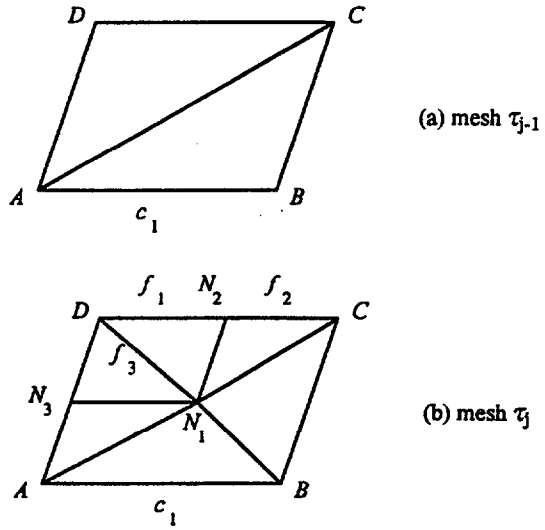


Figure 2: Example of refinement.

We should stress here that the finest mesh is changing at derefining because during the derefinement procedure all levels of meshes are managed. An element will be able to be eliminated if and only if at derefining its successors have previously eliminated.

In order to a well understanding of the data structure used in the readaptive process we have to observe that each element, face or node in any level of mesh has only one global number. Let *NUMN*, *NUMF* and *NUMEL* be the total numbers of different nodes, faces and elements respectively, in the sequence T plus the ones introduced in the sack vectors (the sack vectors are defined later). Let *NUMP* be the total number of nodes in the sequence T. For each level of mesh $\tau_j \in T$ the real numbers of nodes, edges and elements are known, named *NN(j)*, *NF(j)* and *NE(j)*, where $1 \le j \le n$ and n is the number of levels in our structure T.

The data structure that it is proposed to implement our derefinement algorithm can be summarized as follows:
a) *Structure vectors*, or also called *intermediate vectors*. These vectors are: *IMNODE(1:NUMP)*, for the nodes; *IMFACE(1:ISPF)* for the faces and *IMELEM(1:ISPE)* for the elements. In *IMNODE* only the proper nodes of each level are kept because if one node belongs to a particular mesh, it belongs to the following meshes as well, so *NUMP=NN(n)*. With respect to the faces and elements we can say that *IMFACE* and *IMELEM* keep the global numbers of all faces and elements respectively of each level of the sequence T. For this reason, we can write that

$$ISPF = \sum_{j=1}^{n} NF(j) \quad \text{and} \quad ISPE = \sum_{j=1}^{n} NE(j).$$

b) *Genealogy vectors*: *IR[1:3,1:NUMF]* for the faces and *IXH[1:6,1:NUMEL]* for the elements. For each edge, *IR* reports the numbers of its son edges and its father edge. Similarly, for each element, *IXH* give us the number of its son elements, its father element and the local number of its longest side.
c) *Level vectors*, or also called *derefinement indicators*: *NODES(1:NUMN)*, for the nodes; *NFACES(1:NUMF)* for the edges and *NELES(1:NUMEL)* for the elements. For each one these vectors give us the level in which this node, edge or element is proper and, at the same time, the sign of them is used to control the derefinement procedure.
d) *Sack vectors: NNSAC, NFSAC* and *NESAC*. In these vectors, the global number of nodes, edges and elements eliminated are kept to be used in next refinements. These vectors enable us to do not have to renumber the nodes, faces and elements after each derefinement application.
e) Finally, and in order to facilitate the computation of the derefinement condition for each node, we introduce a new vector, said *IEX(1:NUMN)*. For each node *IEX* reports the *surrounding edge*, that is the number of the edge in which that node is in the middle point. Because, as derefinement condition the following one has been used: the absolute difference, between the numerical solution at one node and the interpolated solution of the ends of its surrounding edge, is checked. If this absolute difference is less than a constant -that can be fixed for each program run-, the node can be canceled. This imposed constant will be called epsilon. However, the relative difference can also be used. If another derefinement condition is applied, this vector can be ignored in the data structure..

2.3 The derefinement procedure

Let $T = \{ \tau_1 < \tau_2 < \ldots < \tau_n \}$ be a sequence of structured triangular grids and we want to obtain another sequence after derefining T, said T'. This means that the new sequence can be written as follows:
$T' = \{ \tau_1 < \tau_2' < \ldots < \tau_m' \}$ where $m \le n$.

The derefinement algorithm can be described shortly in this form:

INPUT: Sequence $T = \{ \tau_1 < \tau_2 < \ldots < \tau_n \}$
Loop in levels of T; For j = n to 2, do:

1. For each proper node of τ_j the derefinement condition is evaluated and the nodes and edges able to be eliminated are pointing out using the derefinement indicators.

2. Conformity of the arising new level j is assured minimizing the derefinement area.

3. Following question is presented:

If some proper node of τ_j stay then,

3.a. New nodal connections are defined for the new level j, said τ_j^j. Genealogy vectors of τ_j^j and τ_{j-1} are modified.

In other case,

3.b. The current level j is deleted in the structure vectors. Genealogy vectors of τ_{j-1} are modified.

4. The changes in the mesh are inherited to the following meshes. The structure vectors are compressed.

5. A new sequence of nested meshes T^j is obtained. This sequence is the new input in the next iteration of the loop on meshes. $T^j = \{ \tau_1 < \tau_2 < ... < \tau_{j-1} < \tau_j^j < ... < \tau_{n_j}^j \}$. Being $n_{j+1}-1 \le n_j \le n_{j+1}$ and $n_{n+1}=n$.

OUTPUT: Sequence $T' = \{ \tau_1 < \tau_2' < ... < \tau_m' \}$

Some comments about the derefinement algorithm can be remarked:

The derefinement sequence of T is obtained directly at the end of the loop in levels of T, that means $T'=T^2$.

It is worth to be noted only proper nodes are eligible in each mesh-level, and out of these, only those suitable to be canceled out are taken for evaluation. That is because if *N* is a particular node that it cannot be canceled and *c* is its surrounding edges, then any node of the elements in which *c* is an edge cannot be canceled either. In this manner we can assure that the meshes of the new sequence are nested. An example of this question can be observed in Fig. 2: if N_1 has to stay, nodes A, B, C and D will also stay. This allows us to evaluate the derefinement condition in a minimum number of nodes, and only once for each of those.

The conformity of all meshes in each sequence is assured to maintaining some nodes that otherwise, and concerning the derefinement condition, could have been canceled. In fact, if a node belongs to the longest edge of an element in which in other edge there is another node, the node remains. For instance, in Fig. 2 , if N_2 or N_3 remains, N_1 remains too.

In the derefinement process some new elements may arise. For instance, in Fig. 2, if node N_3 is eliminated, the element AN_1D appear. The global number of the element AN_1N_3 is assigned to the new element and the global number of the element N_1N_3D is kept in the vector *NESAC*. This example remarks the flexibility of the derefinement algorithm.

Once it is obtained T', the number of equation associated to each degree of freedom/node must be redefined, maintaining the global number of the node and the previous numerical solution. At the same time the new number of equations of each level is calculated. This is important to apply again the finite element solver.

3. CONVECTION-DIFFUSION PROBLEM

In this section the principal numerical aspects about an implicit formulation of the evolutive convection-diffusion problem studied in (ref. 9) are presented. After, in the following section we will apply the refinement/ derefinement algorithm to solve a model problem, using this numerical formulation.

3.1 Implicit formulation

We consider the convection-diffusion problem defined in a two-dimensional domain Ω, of boundary Γ:

$$\frac{\partial u}{\partial t} + \bar{v}\cdot\bar{\nabla}u - \bar{\nabla}\cdot\left(k\bar{\nabla}u\right) = f \qquad (1)$$

where $u = u(\bar{x},t)$ is the solution in a fluid element placed in $\bar{x} = x_1\vec{i} + x_2\vec{j}$ at time t; fluid velocity is given by $\bar{v} = \bar{v}(\bar{x})$; we study the lineal model, $k = k(\bar{x})$; $f = f(\bar{x},t)$ are the external sources. We suppose boundary and initial conditions such that uniqueness and existence of solution are assured. Equation (1) can be written as:

$$\frac{du}{dt} - \bar{\nabla}\cdot\left(k\bar{\nabla}u\right) = f \qquad (2)$$

Let be a fluid element at point $\underline{P}$, defined by $\underline{\bar{x}}$, at time t_n. After a time step Δt, this fluid element will be in the position P, defined by $\bar{x}$, at time t_{n+1}; such that $t_{n+1} = t_n + \Delta t$ and $\bar{x} = \underline{\bar{x}} + \Delta\bar{x}$. Using the following approximation

$$\frac{du}{dt} \approx \frac{u(\bar{x},t_{n+1}) - u(\underline{\bar{x}},t_n)}{\Delta t} = \frac{u^{n+1}(\bar{x}) - u^n(\underline{\bar{x}})}{\Delta t} \qquad (3)$$

to do an Euler implicit approximation in (2), we obtain:

$$u^{n+1}(\bar{x}) - \Delta t\,\bar{\nabla}\cdot\left[k(\bar{x})\bar{\nabla}u^{n+1}(\bar{x})\right] = \Delta t f^{n+1}(\bar{x}) + u^n(\underline{\bar{x}}) \qquad (4)$$

In (4) we should have the problem to evaluate $u^n(\underline{\bar{x}})$ in the discrete domain. In order to solve this problem, we try to write the equation (4) depending only of what happen at the point P at every time. For this reason we begin approximating $\underline{\bar{x}}$. For all $i = 1,2$:

$$\underline{x}_i = x_i(t_{n+1} - \Delta t) = x_i - v_i(\bar{x})\Delta t + \frac{\Delta t^2}{2}\bar{v}(\bar{x})\cdot\bar{\nabla}v_i(\bar{x}) + 0(\Delta t^3)$$

From this equation we can obtain $\Delta x_i = x_i - \underline{x}_i$ to be used in the development that approximate $u^n(\underline{\bar{x}})$:

$$u^n(\underline{\bar{x}}) = u^n(\bar{x} - \Delta\bar{x}) = u^n(\bar{x}) - \sum_{i=1}^{2}\Delta x_i\frac{\partial u^n(\bar{x})}{\partial x_i} + \\ + \frac{1}{2}\left[2\Delta x_1\Delta x_2\frac{\partial^2 u^n(\bar{x})}{\partial x_1\partial x_2} + \sum_{i=1}^{2}\Delta x_i^2\frac{\partial^2 u^n(\bar{x})}{\partial x_i^2}\right] + 0(\|\Delta\bar{x}\|^3)$$

and then,

$$u^n(\underline{\bar{x}}) = u^n(\bar{x}) - \Delta t\sum_{i=1}^{2}v_i(\bar{x})\frac{\partial u^n(\bar{x})}{\partial x_i} + \frac{\Delta t^2}{2}\sum_{i=1}^{2}\left[\bar{v}(\bar{x})\cdot\bar{\nabla}v_i(\bar{x})\right] + \\ + \frac{\Delta t^2}{2}\sum_{i=1}^{2}\sum_{j=1}^{2}v_i(\bar{x})v_j(\bar{x})\frac{\partial^2 u^n(\bar{x})}{\partial x_i\partial x_j} + 0(\Delta t^3)$$

Finally, if we introduce this last expression in (4) we get the following implicit formulation that approximate the evolutive convection-diffusion process, where all the terms are evaluated at the same point P, defined by $\bar{x}$:

$$u^{n+1} - \Delta t\,\bar{\nabla}\cdot\left[k\bar{\nabla}u^{n+1}\right] = \Delta t f^{n+1} + u^n - \Delta t\,\bar{v}\cdot\bar{\nabla}u^n + \\ + \frac{\Delta t^2}{2}\sum_i(\bar{v}\cdot\bar{\nabla}v_i)\frac{\partial u^n}{\partial x_i} + \frac{\Delta t^2}{2}\sum_{i,j}v_iv_j\frac{\partial^2 u^n}{\partial x_i\partial x_j} \qquad (5)$$

Now, considering boundary conditions, it is easy to obtain the variational formulation and then apply the finite element method. We have used a consistent integration in all terms of the final formulation.

3.2 Stability

Concerning the former formulation, we present now the principal stability results obtained in one and two dimension by the Von Neuman method. These results have been analyzed in (ref. 9).

In one dimension, we consider a regular mesh with elements of diameter h, so if we call $C=v\Delta t/h$ to the Courant number and $P_e=vh/k$ to the Peclet number we have the following stability condition:

$$C \le \left[\frac{1}{P_e^2} + \frac{1}{3}\right]^{1/2} + \frac{1}{P_e} \qquad (6)$$

A general stability analysis in two dimensions it would be so much complex. For this reason, we take a particular case with a mesh of rectangular triangles which hypotenuses are oriented on the $\pi/4$ direction and their longitudes are $h\sqrt{2}$. We suppose a velocity field with a direction defined by the angle α with the x_1 axis. We study only two directions of wave propagation: x_1 axis and $3\pi/4$. In this cases we obtain the following stability conditions, given by (7) and (8) respectively:

$$C \le \frac{1}{\cos^2\alpha}\left[\left(\frac{1}{P_e^2} + \frac{1}{3}\cos^2\alpha\right)^{1/2} + \frac{1}{P_e}\right] \qquad (7)$$

$$C \le \frac{1}{1-\mathrm{sen}2\alpha}\left[\left(\frac{4}{P_e^2} + \frac{1}{3}(1-\mathrm{sen}2\alpha)\right)^{1/2} + \frac{2}{P_e}\right] \qquad (8)$$

In both cases, the worst condition results when the directions of the velocity field and wave propagation are the same; $\alpha=0$ in (7) and $\alpha=3\pi/4$ in (8). In this case, condition given by (8) results worse than the one given by (7). We can see that, in the hypothesis of same direction of velocity field and wave propagation, the conditions given by (7) and (8) can be obtained from the condition (6) only taking as h the maximum distance between two points of the two-dimensional element on this direction.

3.3 Consistency

We analyze the consistency of the implicit formulation using a similar method that in (ref. 9-10). That is by comparing the improvement, after one time step, of the numerical and the exact solutions. For the analyzed implicit formulation, particularized in one dimension and with constant coefficients and without sources in the second member, it can be proved that it is globally second-order accurate; that means:

$$G(\eta) = G_e(\eta) + 0(\eta^3) \qquad (9)$$

where $\eta=\xi h$, being ξ the wave number; $G(\eta)$ and $G_e(\eta)$ are the improvement after one time step, in a generic node of a regular mesh, of the numerical and the exact solution, respectively.

3.4 Adaptive strategy

We have solved the convection-diffusion problem using several error indicators for the local refinement (ref. 9). In this paper we have introduced the new following error indicator, ε_i, for an element Ω_i:

$$\varepsilon_i = h_i^2\left|\nabla u_h\right| \qquad (10)$$

being h_i the diameter of Ω_i and u_h the lineal numerical

solution in the triangular element. This error indicator is a measure of the maxim variation of u_h in Ω_i pondered with its diameter h_i, in order to do not refine excessively in the high gradient areas.

4. NUMERICAL RESULTS

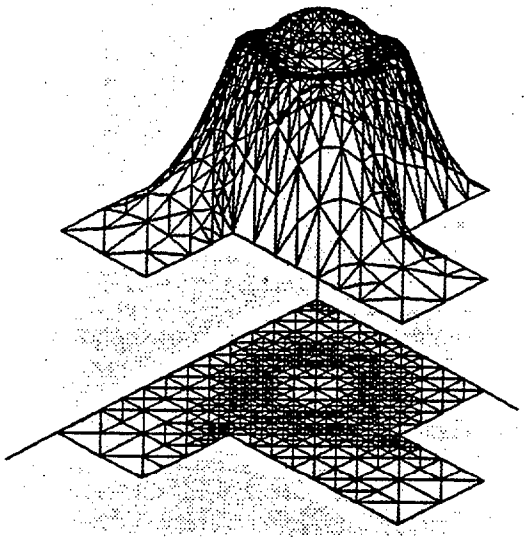
4.1 A Quasi-evolutive Problem

Figure 3 shows some meshes and the solutions in different time of events of a Poisson problem in which the function f of the second member is time-dependent:

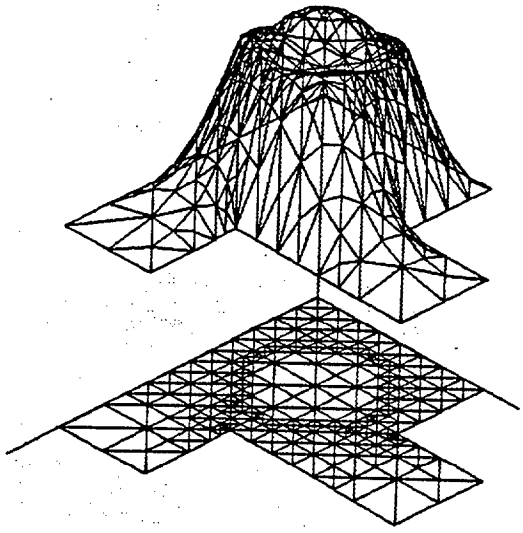
$$f(x_1,x_2)=\begin{cases}0 & \text{if} \quad d>r_1(t)\\ 50 & \text{if} \quad r_2(t)<d<r_1(t)\\ -5 & \text{if} \quad d<r_2(t)\end{cases}$$

where $r_1(t)=0.1(1+t)$, $r_2(t)=0.1t$ and d is the distance between (x_1,x_2) and (0.5,0.5).

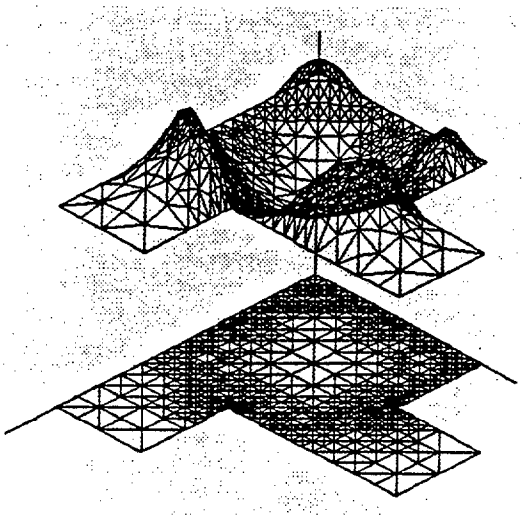
The domain geometry is non symmetrical and we have Dirichlet condition u=1 in the boundary.



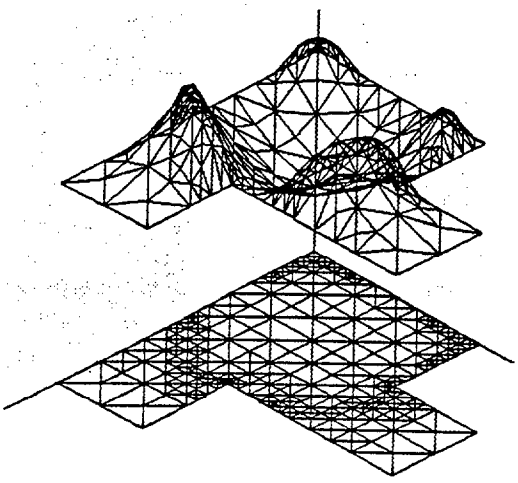
t=3, 528 nodes



t=3, 257 nodes



t=5, 768 nodes



t=5, 287 nodes

Figure 3: Refined meshes and numerical solutions on the left. Respective derefined meshes and new approximations on the right.

As error indicator for the local refinement we apply one proposed for elliptic problems in (ref. 1-2) and parameter γ = 0.3. Parameter epsilon used was 0.013 with absolute difference as derefinement condition. Finally, at each time event one refinement followed by the derefinement procedure has been used as readaptive strategy.

4.2 A Convection-diffusion Problem

In order to show the efficiency of the readaptive process combined with the implicit formulation presented in Section 3 of this paper, we consider the following model problem.

Let Ω be an unit square domain centered at the point (0.5,0.5), with null Neumann conditions on its boundary Γ. We study the problem with a revolving velocity field with $v_1 = \omega x_1(1-x_1)(x_2-0.5)$ and $v_2 = \omega x_2(0.5-x_1)(1-x_2)$ such that $\vec{\nabla}\cdot\vec{v}=0$ and it is tangent to Γ. The maximum value of the velocity field is $\omega/8$ and it is taken in the middle points of the sides of Ω. In the present application we have chosen $\omega = 2.5\cdot 10^4$. The value of the diffusion coefficient is $k = 0.1$. That is, a significance Peclet number for the presented problem is $3.125\cdot 10^4$. We suppose none external sources, $f = 0$.

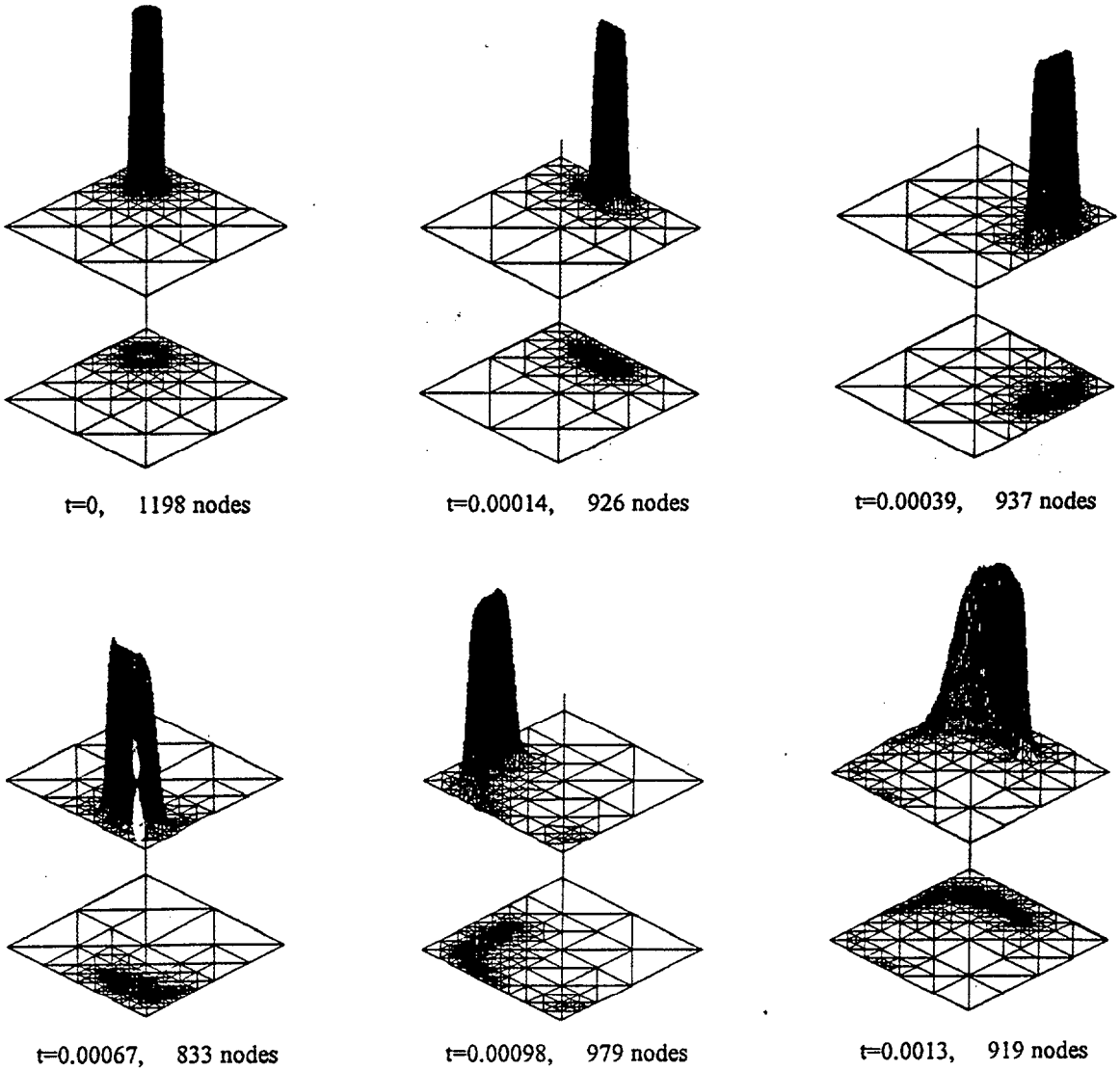


Figure 4: Meshes and numerical solutions for an evolutive convection-diffusion problem at six time events.

The initial solution is a given function that it is approximated in Fig. 4 using a readaptive strategy. This strategy enable us to get a good approximation with a minimum number of nodes.

Error indicator presented in section 3.4 is used for local refinements with parameter $\gamma = 0.9$, and the parameter epsilon at derefining was 0.005.

It is worth to be noted that the number of nodes in the mesh takes practically the same value during the evolutive process.

Until the last time showed in the figure, $t = 0.0013$, 2592 time steps have been realized. The readaptive strategy was: three refinements followed by the derefinement procedure with increment of the time after each refinement or derefinement. For evaluation of the time increment after each refinement or derefinement the stability conditions obtained in section 3.2 has been considered.

5. CONCLUSIONS

The refinement/derefinement combination is very useful to solve time-dependent problems in which moving refinement areas are required. The derefinement algorithm enables us to use the multigrid method in order to solve the equation system associated to the finite-element method. The additional computation time required by this algorithm amounts less than 1% of total execution time.

The implicit formulation proposed for solving the convection-diffusion problem presents good results, moreover if it is combined with a readaptive procedure. Without this readaptive procedure, the cost in CPU time for solving evolutive problems with Peclet numbers such elevated as the ones managed in this paper should be too high.

As another application, the derefinement algorithm can be used for local refining. Derefining after global refinement is thereby equivalent to a local refinement procedure, not requiring the use of error indicators. This implies clear advantages in the class of problems for which error indicators cannot be found or their computation in too costly.

Finally, this algorithm can also be used in order to obtain the best piece-wise support to interpolate a given two-dimensional function, or to approach the initial solution in an evolutive problem.

REFERENCES

1 J.P. de S.R. Gago, D.W. Kelly, O.C. Zienkiewicz and I. Babuska, International J. Numerical Methods in Engineering, 19 (1983) 1621- 1656 .

2 D.W. Kelly, J.P. de S.R. Gago, O.C. Zienkiewicz and I. Babuska, International J. Numerical Methods in Engineering, 19 (1983) 1593-1619.

3 J.Z. Zhu and O.C. Zienkiewicz, J. Computational in Applied Numerical Methods, 4 (1998) 197-204.

4 M.C. Rivara, International J. Numerical Methods in Engineering, 24 (1987) 1343-1354.

5 M.C. Rivara, International J. Numerical Methods in Engineering, 28 (1989) 2889-2906.

6 A. Plaza, L. Ferragut and R. Montenegro, in J. van Leeuwen (Ed.), XIIth World Computer Congress IFIP'92, Madrid, Spain, September 4-7, 1992, Elsevier Science Publishers, 1992, to appear.

7 W. Hackbush and V. Trottengurg (Ed.), Multigrid Methods, Lectures Notes in Mathematics, Springer-Verlag, Berlin, 1982.

8 W. Hackbush and V. Trottengurg (Ed.), Multigrid Methods II, Lectures Notes in Mathematics, Springer-Verlag, Berlin, 1982.

9 R. Montenegro, G. Montero, L. Ferragut and G. Winter, Revista Internacional de Metodos Numéricos para Calculo y Diseño en Ingeniería, 5 (1989) 535-560.

10 J. Peraire, O.C. Zienkiewicz and K. Morgan, International J. Numerical Methods in Engineering, 22 (1986) 547-574.

Referencias

[AhHoU83] A.V. Aho, J.E. Hopcroft y J.D. Ullman, *Estructuras de datos y algoritmos*, versión en castellano en Ed. Addison-Wesley Iberoamericana, 1988.

[AlLiM92] L. Alvarez, P.-L. Lions and J.-M. Morel, 'Image selective smoothing and edge detection by nonlinear diffusion. II`, SIAM *J. Num. Anal.*, **29**, núm. 3, 845-866, Junio- 1992.

[AraGe87] R. Aravind and A. Gersho,'Image compression based on vector quantization with finite memory`, *Optical Engineering,* **26,** 570-580, Julio-1987.

[BabRh78] I. Babuska and W. Rheinboldt, 'Error estimates for adaptive finite element computations`, SIAM *J. Numer. Anal.* **15,** 736-754 (1978).

[Balca92] J. L. Balcázar, *Eficiència dels Algoritmes,* Universitat Politècnica de Catalunya, 1992.

[BanSh80] R.E. Bank, and A. Sherman, 'A refinement algorithm and dinamic data structure for finite element meshes`, *Technical Report 166,* University of Texas Center for Numerical Analysis, 1980.

[BanSh83] R.E. Bank, A.H. Sherman and A. Weiser, 'Refinement algorithms and data structures for regular local mesh refinement`, in R. Stepleman et al. (eds.) *Scientific Computing*, IMACS, North-Holland, Amsterdam, 1983, pp. 3-17.

[BraBr87] G. Brassard y P. Bratley, *Algoritmica: Concepción y análisis,* versión en castellano Ed. Masson, 1990.

[Ciarl78] P.G. Ciarlet, *The finite element method for elliptic problems,*in J.L. Lions et al. (eds.) *Studies in Mathematics and its applications* **4**, North-Holland, Amsterdam, 1983.

[Cross90] J. T. Cross, 'A two-dimensional triangular mesh generator using the advanced front method`, *Technical Report CR/662/91,* Department of Civil Engineering, University College Swansea, U. K., 1990.

[Cuest92] P. D. Cuesta, 'Contribución a la discretización de dominios planos no convexos mediante algoritmos de triangulación y técnicas de regularización de mallas`, *Tesis Doctoral,* Departamento de Matemáticas, Universidad de Las Palmas de Gran Canaria, 1992.

[Diaz87] J. Díaz, *Complejidad Concreta y Complejidad Abstracta de Algoritmos: Un panorama actual.* IV Escuela de Verano de Informática, A.E.I.A., 1982.

[FeMoW91] L. Ferragut, R. Montenegro, G. Winter y A. Nuñez, 'Accurate extraction of interconnect capacitances by adaptive mixed finite element method', *The Euromicro Journal, Microprocessing and Microprogramming,* **32,** 61-68, North-Holland (1991).

[FePlM87] L. Ferragut, A. Plaza and R. Montenegro, 'Readaptive processes of nested meshes: refinement/derefinement', comunicación en el International Congress on Extrapolation and Rational Approximation, Puerto de la Cruz (Tenerife), 1992.

[FePlM91] L. Ferragut, A. Plaza y R. Montenegro, 'Procesos readaptativos de mallas estructuradas: refinamiento/desrefinamiento', *Actas* XII C.E.D.Y.A., II Congreso de Matemática Aplicada, 453-459, 1991.

[Ferra87a] L. Ferragut, 'Una solucion al problema de la programación de métodos de elementos finitos autoadaptativos', Anales Ing. Mec. **5**, 201-206 (1987).

[Ferra87b] L. Ferragut, 'Neptuno, un sistema de elementos finitos autoadaptativo' (en Fortran), Departmento de Matemáticas Aplicadas y Métodos Informáticos, Madrid (1987).

[GaKeZ83] J.P. de S.R. Gago, D.W. Kelly, O.C. Zienkiewicz and I. Babuska, 'A posteriory error analysis and adaptive processes in the finite element method: Part II - Adaptive mesh refinement', *Int. J. Num. Meth. Eng.* **19,** 1621- 1656 (1983).

[GerYa85] A. Gersho and M. Yano, 'Adaptive vector quantization by progressive codevector replacement', IEEE, 133-136, 1985.

[HacTr82] W. Hackbush and V. Trottengurg (eds.), *Multigrid Methods, Lectures Notes in Mathematics*, Springer-Verlag, Berlin, 1982.

[HacTr86] W. Hackbush and V. Trottengurg (eds.), *Multigrid Methods II, Lectures Notes in Mathematics*, Springer-Verlag, Berlin, 1986.

[JoSuK91] B. Johnston, J. Sullivan and A. Kwasnik, 'Automatic conversion of triangular finite element meshes to quadrilateral elements', *Int. j. numer. methods eng.* **31**, 67-84 (1991).

[KeGaZ83] D.W. Kelly, J.P. de S.R. Gago, O.C. Zienkiewicz and I. Babuska, 'A posteriory error analysis and adaptive processes in the finite element method: Part I - Error analysis', *Int. J. Num. Meth. Eng.* **19**, 1593-1619 (1983).

[LeHuU91] R. W. Lewis, H.C. Huang, A.S. Usmani and J.T. Cross,'Finite element analysis of heat transfer and flow problems using adaptive remeshing including application to solidification problems', *Int. J. Num. Meth. Eng.* **32**, 767-781 (1991).

[Marr82] D. Marr, *Vision,* W. H. Freemann (ed.), S. Francisco, CA, 1982.

[MoMoW89] R. Montenegro, G. Montero, G. Winter y L. Ferragut, 'Aplicación de métodos finitos adaptativos a problemas de convección-difusión en 2-D", *Revista Int. Métodos Numéricos para el Cálculo y Diseño en Ingeniería* **5**, 535-560 (1989).

[Monte89] R. Montenegro, *Aplicación de métodos de elementos finitos adaptativos a problemas de convección-difusión,* Tesis Doctoral, E.T.S.I. Industriales, Universidad de Las Palmas, 1989.

[MoMoW90] G. Montero, R. Montenegro, G. Winter and L. Ferragut, 'Aplicación de esquemas EBE en procesos adaptativos', *Rev. Int. Met. Num. Cal. Dis. Ing.* **6**, 311-332 (1990).

[Netra77] A.N. Netravali, 'Interpolative picture coding using a subjective criterion', IEEE. *Trans. Commun.* COM-25 (5), 503 (1977).

[PerPe93] J. Peraire and J. Peiro, 'Multigrid solution of the 3-D compressible Euler Equations on unstructured tetrahedral grids', *Int. J. Num. Meth. Eng.,* **36**, 1029-1044 (1993).

[PeVaM87] J. Peraire, M. Vahdati, K. Morgan and O. Zienkiewicz, 'Adaptive remeshing for compressible flow computations', *J. Comp. Phys.,* **72**, 449-466 (1987).

[PeZiM86] J. Peraire, O. Zienkiewicz and K. Morgan, 'Shallow water problems: a general explicit formulation', *Int. J. Num. Meth. Eng.,* **22**, 547-574 (1986).

[PlFeM92] A. Plaza, L. Ferragut and R. Montenegro, 'Derefinement algorithms of nested meshes', in J. van Leeuwen (ed.) *Algorithms, Software, Architecture,* IFIP, Elsevier Science Publishers B.V. (North-Holland), Amsterdam, 1992, pp. 409-415.

[PlMoF92] A. Plaza, R. Montenegro and L. Ferragut, 'An adaptive refinement/derefinement algorithm of structured grids for solving time-dependent problems', in Ch. Hirsch et al. (eds.), *Numerical Methods in Engineering*, Elsevier Science Publishers B.V., Amsterdam, 1992, pp. 225-232.

[Rivar84a] M.C. Rivara, 'Algorithms for refining triangular grids suitable for adaptive and multigrid techniques', *Int. J. Num. Meth. Eng.* **20**, 745-756 (1984).

[Rivar84b] M.C. Rivara, 'Mesh refinement processes based on the generalized bisection of simplices', SIAM *J. Num. Anal.* **21**, 604-613 (1984).

[Rivar84c] M.C. Rivara, 'A dinamic multigrid algorithm suitable for partial differential equations with singular solutions', in A. Balakrishnan and E. Polak (eds.), *Proceedings IFIP Working Conference 1984.* Santiago, Chile, *Lesture Notes in Control and Information Sciences,* 1986, Cap. 20, 359-370.

[Rivar87] M.C. Rivara, 'A grid generator based on 4-triangles conforming. Mesh-refinement algorithms', *Int. J. Num. Meth. Eng.* **24**, 1343-1354 (1987).

[Rivar89] M.C. Rivara, 'Selective refinement/derefinement algorithms for sequences nested triangulations', *Int. J. Num. Meth. Eng.* **28**, 2889-2906 (1989).

[RheMe80] W. C. Rheinboldt and C. K. Mesztenyi, 'On a data structure for adaptive finite element mesh refinements', *ACM Transactions on Mathematical Software,* **6**, 166-187, (1980).

[RosSt75] I. G. Rosenberg and F. Stenberg, 'A lower bound on the angles of the triangles constructed by bisecting the longest side', *Mah. Comp.* **29**, 390-395 (1975).

[SheWe91] M.S. Shephard and N.P. Weatheerill, 'Preface', *International Numerical Methods in Engineering.* Special Issue on Adaptive Meshing, **32**, 651 (1991).

[Styne80] M. Stynes, 'On faster convergence of the bisection method for all triangles', *Mah. Comp.* **35**, 1195-1201 (1980).

[Thack84] W. C. Thacker, 'A brief review of techniques for generating irregular computational grids', *Int. J. Num. Meth. Eng.* **15**, 1335-1341 (1980).

[ZhuZi88] J.Z. Zhu and O.C. Zienkiewicz, ´Adaptive techniques in the finite element method`, *J. Comp. Appl. Num. Meth.* **4**, 197-204 (1988).

[ZhuZi91] J. Z. Zhu and O. C. Zienkiewicz, ´A new approach to the development of automatic quadrilateral mesh generation`, *Int. j. numer. methods eng.* **32**, 783-810 (1991).

[Zienk82] O. C. Zienkiewicz, *El método de los elementos finitos,* Ed. Reverté, 1982.

[ZieZh87] O. C. Zienkiewicz and J. Z. Zhu, ´A simple error estimator and adaptive procedure for practical engineering analysis`, *Int. j. numer. methods eng.* **24**, 337-357 (1987).

[ZieZh91] O.C. Zienkiewicz and J. Z. Zhu, ´Adaptivity and mesh generation`, *Inter. J. Num. Meth. Eng.* **32**, 783-810 (1991).

Universidad de Las Palmas de Gran Canaria
UNIDAD DE TERCER CICLO Y POSTGRADO

C/. Pérez del Toro, 5
35004 Las Palmas de Gran Canaria

D. PEDRO ALMEIDA BENITEZ, DIRECTOR DE ESTUDIOS DE TERCER CICLO Y POSTGRADO DE LA UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA.

C E R T I F I C A: *Que el* ***DR. D. FRANCISCO RUBIO ROYO,*** *ha actuado como* ***VOCAL*** *del Tribunal de Lectura de la Tesis Doctoral* ***"ALGORITMOS DE DESREFINAMIENTO EN MALLADOS ESTRUCTURADOS BIDIMENSIONALES",*** *realizada en el Departamento de Matemáticas, por* ***D. ANGEL PLAZA DE LA HOZ,*** *cuyo acto de mantenimiento y defensa ha tenido lugar en la Facultad de Informática, el día 16 de Junio de 1.993, a las 13'00 Horas.----------*

Y para que así conste y surta los efectos oportunos a petición del interesado, expido el presente en Las Palmas de Gran Canaria, a dieciseis de Junio de mil novecientos noventa y tres.----------

---

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA • UNIDAD DE TERCER CICLO Y POSTGRADO