UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

DEPARTAMENTO DE MATEMÁTICAS

TESIS DOCTORAL

GENERACIÓN DE MALLAS TRIDIMENSIONALES MEDIANTE LA TRIANGULACIÓN DE DELAUNY

JOSE M. ESCOBAR SÁNCHEZ

Las Palmas de Gran Canaria, marzo de 1995

Título de la tesis:

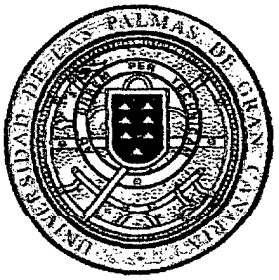
GENERACIÓN DE MALLAS TRIDIMENSIONALES MEDIANTE LA TRIANGULACIÓN DE DELAUNY

Thesis title:

GENERATION OF THREE-DIMENSIONAL MESHES BY MEANS OF THE TRIANGULATION OF DELAUNY

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

DEPARTAMENTO DE MATEMÁTICAS



TESIS DOCTORAL

GENERACIÓN DE MALLAS TRIDIMENSIONALES MEDIANTE LA TRIANGULACIÓN DE DELAUNAY

JOSÉ MARÍA ESCOBAR SÁNCHEZ

Las Palmas de Gran Canaria, 1995

24/1994-95

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
UNIDAD DE TERCER CICLO Y POSTGRADO

Reunido el día de la fecha, el Tribunal nombrado por el Excmo. Sr. Rector Magfco. de esta Universidad, el aspirante expuso esta TESIS DOCTORAL.

Terminada la lectura y contestadas por el Doctorando las objeciones formuladas por los señores jueces del Tribunal, éste calificó dicho trabajo con la nota de APTO CUM LAUDE POR UNANIMIDAD

Las Palmas de G. C., a 31 de marzo de 1995.

El Presidente: Dr. D. Gabriel Winter Althaus,

El Secretario: Dr. D. Angel Plaza de la Hoz,

Angel Plaza

El Vocal: Dr. D. Luis Ferragut Canals,

El Vocal: Dr. D. Francisco Serón Arbeloa,

El Vocal: Dr. D. Francisco Javier Elorza Tenreiro,

El Doctorando: D. José Mª Escobar Sánchez,

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

DEPARTAMENTO DE MATEMÁTICAS

PROGRAMA DE DOCTORADO:
MÉTODO DE LOS ELEMENTOS FINITOS EN INGENIERÍA

TESIS DOCTORAL

GENERACIÓN DE MALLAS TRIDIMENSIONALES MEDIANTE LA TRIANGULACIÓN DE DELAUNAY

José María Escobar Sánchez

1995

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

DOCTORADO EN MATEMÁTICAS

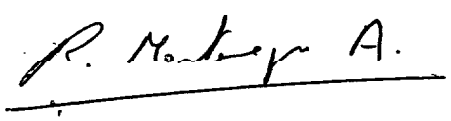
PROGRAMA DE DOCTORADO:
MÉTODO DE LOS ELEMENTOS FINITOS EN INGENIERÍA

GENERACIÓN DE MALLAS TRIDIMENSIONALES MEDIANTE LA TRIANGULACIÓN DE DELAUNAY

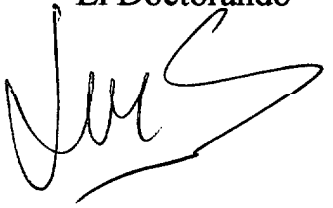
Tesis doctoral presentada por José María Escobar Sánchez

Dirigida por el Dr. D. Rafael Montenegro Armas

El Director



El Doctorando



Las Palmas de Gran Canaria, a 8 de Febrero de 1995.

ABSTRACT

Delaunay triangulation in dimension two and three is one of the most used methods in non-structured mesh generation nowadays. It is very suitable for the application of the finite elements method, due to its geometrical and algorithmical characteristics. The main aim of this Thesis is to develop a program, based on Delaunay triangulation, for the automatic generation of three-dimensional meshes and able to solve efficiently some intrinsic problems of the triangulation method.

The first section in this Thesis is an introductory one. The fundamental concepts and properties of Delaunay triangulation, and its connections with Voronoi diagram are put forward.

Among all the existing algorithms, it has been chosen one based on the well-known Watson algorithm. This is an incremental type algorithm, that is, it builds Delaunay triangulation by adding points one by one. The advantage offered by that kind of algorithms lies in their ability to adapt themselves to a local refinement process of the mesh. The performance of this algorithm, whose main ideas are put forward in the first section, causes some problems when the points are not placed in general position. When the algorithm is put into practice, these problems get bigger due to round-off errors made by the computer when it works with float points.

The second section centers on the details and modifications of the algorithm. Thus, it is showed the date structure that stands the information of the triangulation and the way in which these are stored and renewed as new points are being added. A fundamental part of this algorithm is the one that mentions the modifications carried out to avoid the above mentioned round-off errors.

The definition of polihedric domains and the automatic generation of points are the subjects with which the third section deals. The quality of Delaunay triangulation depends on the relative positions the points have, so that the importance of their

generation in an adequate way. Another aspect taken into account in this section is the conformity of the triangulation with the domain boundary. Although no definitive solutions are given, it is pointed out a procedure based on the addition of new points to the triangulation in the intersection between this last one and the domain. Some promising results have been obtained in the cases studied, although there still remain a few problems.

The forth section deals with the presentation of applications of the mesh generator to several examples. The applications come with several experimental numerical results such as CPU times, measures of the mesh quality, etc.

In the last two sections the conclusions and some open research lines are pointed out, and perspectives for future works are established.

RESUMEN

La triangulación de Delaunay en dimensión dos y tres es uno de los métodos de generación de mallas no estructuradas que más se utiliza en la actualidad. Sus características geométricas y algorítmicas la hacen muy adecuada para la aplicación del método de elementos finitos. El objetivo principal de esta Tesis es construir un programa, basado en la triangulación de Delaunay, para la generación automática de mallas tridimensionales, y que resuelva de forma eficaz algunos problemas intrínsecos al método de triangulación.

El primer capítulo de esta Tesis tiene un carácter introductorio. En él se exponen los conceptos fundamentales y propiedades de la triangulación de Delaunay y sus conexiones con el diagrama de Voronoi.

Entre los numerosos algoritmos existentes, se ha escogido uno basado en el bien conocido algoritmo de Watson. Este algoritmo es de tipo *incremental*, esto es, crea la triangulación de Delaunay por adición de puntos uno a uno. La ventaja que ofrecen los algoritmos de tipo *incremental* estriba en su capacidad de adaptación a un proceso de refinamiento local de la malla. La realización de este algoritmo, cuyas principales ideas son presentadas en el primer capítulo, causa problemas cuando los puntos no están situados en *posición general*. Estos problemas son aún mayores en la práctica debido a los errores de truncamiento o redondeo que comete el ordenador cuando trabaja con *números en coma flotante*.

El segundo capítulo se centra en los detalles y las modificaciones del algoritmo expuesto en la introducción. Se muestra la estructura de datos que soporta la información de la triangulación y la forma en que éstos son almacenados y renovados a medida que se van añadiendo puntos. Una parte fundamental de este algoritmo es la que hace mención a las modificaciones llevadas a cabo para evitar los errores redondeo mencionados anteriormente.

La definición de dominios poliédricos y la generación automática de puntos son las materias de las que trata el tercer capítulo. La calidad de la triangulación de Delaunay depende de las posiciones relativas que ocupen los puntos; de ahí la importancia que tiene el que estos se generen de forma adecuada. Otro aspecto que se tiene en cuenta en este capítulo es la conformidad de la triangulación con la frontera del dominio. Si bien no se dan soluciones definitivas, se apunta un procedimiento basado en la adición de nuevos puntos a la triangulación en las intersecciones entre ésta y el dominio. En los casos tratados se han obtenido unos resultados prometedores aunque subsisten algunos problemas.

El capítulo cuarto está dedicado a la presentación de aplicaciones del mallador a diversos ejemplos. Las aplicaciones están acompañadas de resultados numéricos experimentales como tiempos de CPU, medidas de calidad de la malla, etc.

En los dos últimos capítulos se exponen las conclusiones, algunas líneas de investigación que quedan abiertas y se señalan perspectivas para trabajos futuros.

OBJETIVOS

Los objetivos que nos hemos marcado en esta Tesis son los siguientes:

- Presentar los conceptos fundamentales de la triangulación de Delaunay y del diagrama de Voronoi.

- Exponer los problemas, debidos a disposiciones especiales de los puntos y a errores de redondeo cometidos por el ordenador, que encuentran los algoritmos que generan la triangulación de Delaunay.

- Desarrollar un programa para construir la triangulación tridimensional de Delaunay que, sin operar con aritmética exacta, evite los problemas debidos a errores de redondeo.

- Establecer una entrada de datos para definir dominios tridimensionales poliédricos.

- Desarrollar un programa que genere puntos automáticamente en el interior y en la superficie de dominios poliédricos de manera que la calidad de la triangulación resultante sea buena para la aplicación del MEF. Además, este programa debe ser capaz de eliminar los tetraedros pertenecientes a la envolvente convexa de la nube de puntos y no pertenecientes al dominio.

- Plantear el problema de la conformidad entre la triangulación y la frontera del dominio y exponer algunos resultados obtenidos y encaminados a solucionar este problema.

- Aplicar el mallador a diversos dominios tridimensionales y exponer los resultados numéricos correspondientes.

Agradecimientos

Quiero expresar mi agradecimiento a mi Director de Tesis, Rafael Montenegro, por haberme marcado las líneas de investigación aquí tratadas y por la aportación de muchas de las ideas que han servido como base para el desarrollo de esta Tesis.

Asimismo, agradezco a la ETSIT y al Departamento de Matemáticas los medios técnicos y bibliográficos facilitados para la elaboración de este trabajo.

Gracias también a la empresa AHEMON, S.A., que a través de la FUNDACIÓN UNIVERSITARIA DE LAS PALMAS (GRAN CANARIA), ha financiado en parte este trabajo.

ÍNDICE

1. INTRODUCCIÓN

La mayoría de los fenómenos físicos son formulados por medio de ecuaciones diferenciales en derivadas parciales. Salvo en ciertos casos particulares, la resolución analítica de estas ecuaciones suele resultar imposible. El interés práctico que tiene el conocer soluciones aproximadas de estas ecuaciones, y la rapidez en los cálculos que proporcionan los ordenadores, han hecho que los métodos numéricos cobren gran importancia. En concreto, el método de los elementos finitos (MEF) se ha revelado como un potente método de resolución aproximada de ecuaciones diferenciales. La esencia de este método consiste en el cálculo de la solución de la ecuación en algunos puntos del dominio de interés, denominados nodos. A partir de estos valores se puede calcular la solución en cualquier otro punto mediante el uso de funciones de interpolación. Los cálculos precedentes requieren, como primer paso, que el dominio esté *discretizado* en pequeñas parcelas, los elementos. El conjunto de estas parcelas se denomina malla del dominio. Este preproceso tiene una importancia crucial en la convergencia de las soluciones numéricas y en su aproximación a los verdaderos valores. La malla debe acercarse lo más posible a algunos requisitos generales, tales como:

- Adaptarse lo mejor posible a la forma del objeto.

- Debe haber mayor densidad de elementos en las zonas donde la solución varíe más rápidamente.

- La transición de una zona donde los elementos sean *grandes* a otra donde sean *pequeños* debe ser gradual.

- No debe haber elementos muy *degenerados*. El sentido preciso de la palabra degenerado depende del tipo concreto de elementos de que se trate. Por ejemplo, si los elementos son tetraedros, éstos se deben aproximar lo más posible al tetraedro regular. Así, un tetraedro será degenerado si es muy plano, o muy *puntiagudo*, o si la diferencia entre la esfera circunscrita y la inscrita es muy grande.

Otros requisitos, como la orientación de los elementos, etc., pueden estar motivados por el problema concreto que se pretenda resolver.

Existen dos tipos fundamentales de mallas: estructuradas y no estructuradas. En las mallas estructuradas la conexión entre elementos obedece a un patrón fijo; en las no estructuradas no existe tal patrón, el número de nodos vecinos de uno dado es variable. Las mallas formadas por una transformación de coordenadas, de un cuadrado en dos dimensiones o un cubo en tres, son ejemplos de mallas estructuradas [KunSte93]. En ellas, el nodo de índices (i, j, k) siempre tiene como vecino por la izquierda al $(i-1, j, k)$ y por la derecha al $(i+1, j, k)$, etc.

Las mallas estructuradas tienen ciertas ventajas sobre las no estructuradas. Son más simples, y también más convenientes para su uso en el método de diferencias finitas. Requieren menos memoria para su almacenamiento en el ordenador y es más fácil controlar la forma y el tamaño de los elementos.

La gran desventaja que ofrecen estas mallas es la poca flexibilidad para adaptarse a dominios con geometrías complicadas. Se han desarrollado numerosas técnicas para encontrar las transformaciones de coordenadas adecuadas: transformación conforme, métodos algebraicos, etc. Incluso con estas técnicas es imposible encontrar una transformación que se adapte satisfactoriamente a la forma de un dominio complicado. Existen multitud de problemas formulados en dominios cuya forma no es adecuada para una *discretización* mediante mallas estructuradas. En estos casos es necesario utilizar mallas no estructuradas. Por lo general, los elementos que componen las mallas no estructuradas son símplices, esto es, triángulos en dimensión dos y tetraedros en dimensión tres. La utilización de símplices es debida, entre otras cosas, a su capacidad para llenar el espacio y para adaptarse a los contornos del dominio sin que se produzcan discontinuidades. La malla formada por símplices recibe el nombre de triangulación. Siempre es posible descomponer un dominio poliédrico (poligonal en dimensión dos) en tetraedros (triángulos). El problema está en encontrar una triangulación que se aproxime

lo más posible a los requisitos mencionados anteriormente, o a cualquiera otros que determine el problema a tratar.

Muchas de las medidas de calidad de una triangulación están motivadas por la aplicación del método de los elementos finitos. Estas medidas toman en consideración las longitudes, ángulos y áreas o volúmenes de los triángulos o tetraedros de la triangulación. En ocasiones, la relación entre el círculo (esfera) inscrito y circunscrito a un triángulo (tetraedro) da una buena medida de la calidad del elemento. El *condicionamiento* de las matrices asociadas al MEF depende, entre otros aspectos, del mínimo ángulo de todos los triángulos de la triangulación.

Si el tamaño de los elementos de la malla es uniforme, el error cometido por el MEF suele ser mayor en las regiones donde el gradiente de la solución es mayor. Una forma de disminuir el error consiste en aumentar la densidad de elementos en estas zonas. Esta operación se conoce con el nombre de *refinamiento local*. En problemas evolutivos la localización de la zona donde el gradiente de la solución es mayor puede cambiar con el tiempo. El refinamiento global de toda la malla puede hacer que el número de nodos aumente prohibitivamente. En dimensión dos existen técnicas de *desrefinamiento* que disminuyen la densidad de elementos en las regiones en las que la solución es más uniforme. Sin embargo, en dimensión tres el desrefinamiento está aún en su fase inicial de desarrollo.

La triangulación de Delaunay reúne algunas características que la hacen adecuada para la aplicación del MEF. Dado un conjunto X de puntos de un espacio euclideo de dimensión d, la triangulación de Delaunay crea una triangulación en la que los vértices de los símplices son los puntos de X. Para crear la triangulación de Delaunay existen algoritmos de tipo incremental, como el de Watson, que construyen la triangulación de un conjunto X por adición de puntos uno a uno. Los algoritmos incrementales permiten un refinamiento local de la malla incrementando la densidad de puntos en las zonas en las que se ha cometido más error. Además de la capacidad de refinamiento, la triangulación

de Delaunay posee propiedades geométricas óptimas desde el punto de vista del MEF. De todas las triangulaciones de un conjunto de puntos del plano, la de Delaunay es la que hace máximo el mínimo ángulo de cualquier triángulo. Sin embargo, no existe una generalización de esta propiedad para dimensión mayor que dos, al menos en términos de alguna medida angular de los símplices. En dimensión mayor o igual a dos hay propiedades que hacen referencia al tamaño de las esferas que contienen a los símplices de la triangulación, pero que no nos dan una idea tan clara de la calidad de la triangulación como la que da la propiedad anterior [Fort92]. A pesar de ello, los resultados experimentales revelan que las mallas tridimensionales, construidas a partir de la triangulación de Delaunay, dan buenas medidas de calidad cuando la distribución de puntos es suficientemente regular.

Una de las principales dificultades que plantea la utilización de la triangulación de Delaunay para generar mallas es conseguir que la triangulación *respete* la frontera del dominio de definición, esto es, que no haya ningún tetraedro que corte su superficie. Cuando esto ocurre decimos que la triangulación es conforme con la frontera del dominio. En dimensión dos se resuelve este problema, bien imponiendo que la triangulación contenga todas las aristas que definen el contorno del dominio (*Constrained Delaunay Triangulation*), bien colocando puntos sobre las aristas del dominio en posiciones adecuadas de manera que éstas sean la unión de aristas de la triangulación (*Conforming Delaunay Triangulation*). En dimensión tres hay algoritmos (ver por ejemplo [GeoHec91a] y [WeaHas94]) basados en el intercambio de aristas y caras que consiguen la conformidad con la frontera del dominio. Una vez que se ha definido una malla conforme con la frontera del dominio, una buena estrategia para adaptar la malla, asegurando que en todo momento se mantiene dicha conformidad, sería utilizar métodos de refinamiento/desrefinamiento de mallas encajadas. Esta estrategia se ha desarrollado para triangulaciones bidimensionales [FerMon94] [Plaz93], pero en la actualidad es una línea de investigación abierta en tres dimensiones.

Por otra parte, la realización práctica de algoritmos que generen la triangulación de Delaunay en dimensión d tiene serios problemas cuando los puntos no están en *posición general*, es decir, cuando hay más de $d+1$ puntos sobre la misma d-esfera. Incluso cuando los puntos no están *claramente* en posición general existen problemas debidos a los errores de redondeo que comete el ordenador al trabajar con *números en coma flotante*

1.1. Preliminares.

La generación de mallas es una disciplina que utiliza conceptos de la geometría computacional, y por consiguiente del álgebra lineal y la teoría de poliedros entre otros campos, y cuyas aplicaciones se encuentran principalmente en el método de los elementos finitos. Algunos de los conceptos que se utilizan son definidos de formas diferentes según el marco en el que se encuentren. Resulta conveniente hacer un balance de las nociones más importantes que serán utilizadas más adelante. Las definiciones que exponemos a continuación se encuentran, en su mayor parte, en [GröLov93].

En todo lo sucesivo trabajaremos en un espacio afín euclideo E^d de dimensión d cuyo espacio vectorial asociado es R^d y $\{x_o, \vec{e}_i, 1 \le i \le d\}$ un sistema de referencia construido a partir de un punto fijo $x_o \in E^d$ y la base canónica del espacio vectorial. Denominaremos $\vec{x}$ al vector de posición del punto x respecto del sistema de referencia anterior.

En ocasiones se definirán conjuntos de puntos cuya dimensión es inferior a la del espacio en el que están incluidos. Cuando sea necesario especificar la dimensión antepondremos el prefijo "k-" ($1 \le k \le d$) al nombre del conjunto.

Definición 1: Combinación afín.

Supongamos que $P = \{x_1, x_2, \ldots, x_k\}$ es un conjunto de k puntos de E^d; decimos que el punto x es una combinación afín de los puntos de P si se verifica que:

$$\vec{x} = \sum_{i=1}^{k} \lambda_i \vec{x}_i$$

donde los coeficientes $\lambda_i \in \mathsf{R}$ satisfacen la condición $\sum_{i=1}^{k} \lambda_i = 1$. □

Definición 2: Combinación convexa.

Si x es una combinación afín de los k puntos de P y además se verifica que $\lambda_i \geq 0$ para $i = 1,2,\ldots,k$, decimos que x es una combinación convexa de P. □

Definición 3: Envolvente afín.

Sea $S \subseteq E^d$ un subconjunto no vacío de puntos. El conjunto de todos los puntos de E^d que son combinación afín de algún número finito de puntos de S se denomina envolvente afín de S y se escribe aff(S). En el caso particular de que S sea el conjunto vacío se define aff(S) = $\varnothing$. □

Definición 4: Envolvente convexa.

El conjunto de todos los puntos de E^d que son combinación convexa de algún número finito de puntos de S se denomina envolvente convexa de S y se escribe conv(S). En el caso particular de que S sea el conjunto vacío se define conv(S) = $\varnothing$. □

Definición 5: Independencia afín.

Se dice que un subconjunto de puntos $S \subseteq E^d$ es afinmente independiente si ninguno de sus elementos se puede expresar como combinación afín de los demás elementos de S; en caso contrario diremos que S es afinmente dependiente. □

Como consecuencia de la definición anterior se deduce que cualquier subconjunto afinmente independiente de E^d tiene a lo sumo $d+1$ puntos.

Definición 6: Dimensión afín.

La dimensión afín de un subconjunto $S \subseteq E^d$ es el cardinal del mayor subconjunto de S afinmente independiente menos uno. □

En las definiciones que siguen utilizaremos sistemas de inecuaciones. Para abreviar su escritura pondremos $\vec{a} \leq \vec{b}$ si todas las componentes de $\vec{a}$ son menores o iguales que las correspondientes de $\vec{b}$.

Definición 7: Poliedro convexo.

Sea A una matriz de $l \times d$ elementos y sea $\vec{b}$ un vector de R^l. El conjunto de puntos x tales que $\{\vec{x} \in \mathsf{R}^d / A\vec{x} \leq \vec{b}\}$ es un poliedro convexo. □

Definición 8: Politopo.

Un politopo es cualquier poliedro convexo acotado. Es decir, cualquier poliedro convexo que pueda ser contenido en una bola centrada en el origen y que tenga radio finito mayor que cero. □

Definición 9: Semiespacio.

Sea $\vec{a} \in \mathsf{R}^d - \{\vec{0}\}$ y $a_0 \in \mathsf{R}$, entonces, el poliedro convexo definido por la desigualdad $\{\vec{x} \in \mathsf{R}^d / \vec{a}^T \vec{x} \leq a_0\}$ es un semiespacio. □

Definición 10: Hiperplano.

Sea $\vec{a} \in \mathsf{R}^d - \{\vec{0}\}$ y $a_0 \in \mathsf{R}$, entonces, el poliedro convexo definido por la igualdad $\{\vec{x} \in \mathsf{R}^d / \vec{a}^T \vec{x} = a_0\}$ es un hiperplano. □

De las definiciones anteriores se deduce que cualquier poliedro convexo es la intersección de un número finito de semiespacios. Una inecuación $\vec{a}^T \vec{x} \leq a_0$ se dice que es *válida* respecto a un poliedro P, si P es un subconjunto del conjunto de puntos definido por $\{\vec{x} / \vec{a}^T \vec{x} \leq a_0\}$.

Definición 11: Cara.

Un subconjunto $F \subseteq P$ se dice que es una cara de P si existe una inecuación válida $\vec{a}^T \vec{x} \leq a_0$ respecto a P tal que F es el conjunto de puntos definido por $\{\vec{x} \in P / \vec{a}^T \vec{x} = a_0\}$. □

En particular, el punto v es un vértice del poliedro P si pertenece a P y es una cara del mismo. Un vértice de P es, por tanto, una cara con dimensión afín cero. De

manera similar podemos decir que una arista de P es una cara con dimensión afín uno, etc.

Definición 12: Poliedro.

Dado un conjunto de n politopos Π_i con la misma dimensión afín, se define el poliedro Ω como el conjunto de puntos de E^d dado por:

$$\Omega = \bigcup_{1 \le i \le n} \Pi_i$$

verificándose que cualquiera que sea $\Pi_i \in \Omega$ existe otro politopo diferente $\Pi_j \in \Omega$ tal que la intersección entre ambos es una cara de dimensión afín una unidad inferior a la dimensión de los politopos. □

Definición 13: Símplice.

Un conjunto K de puntos de E^d se dice que es un k-símplice $(1 \le k \le d)$ si $K = \text{conv}(V)$, siendo V es un conjunto de $k+1$ puntos afinmente independientes. Si k coincide con la dimensión del espacio, d, diremos que K es, simplemente, un símplice. □

En un espacio tridimensional los 1-símplices son segmentos, los 2-símplices son triángulos y los 3-símplices, o exclusivamente símplices, son los tetraedros. La orientación de un símplice está dada por el signo del determinante $\det(K)$. Diremos que es positiva si $\det(K) > 0$ y negativa en caso contrario.

$$\det(K) = \begin{vmatrix} x_2^1 - x_1^1 & \cdots & x_{d+1}^1 - x_1^1 \\ \cdots & \cdots & \cdots \\ x_2^d - x_1^d & \cdots & x_{d+1}^d - x_1^d \end{vmatrix}$$

En la definición anterior el número x_i^j es la coordenada j-ésima del punto i-ésimo. Puede demostrarse que el volumen de K es $|\det(K)|/d!$. Si los puntos $x_1, \cdots, x_{d+1}$ no fueran afinmente independientes no definirían un símplice; en este caso el determinante sería nulo. A veces se dice, "abusando del lenguaje", que el símplice está *degenerado*.

La descripción del diagrama de Voronoi y la triangulación de Delaunay como particiones del espacio E^d necesitan definir el concepto de celda.

Definición 14: Celda.

Una *k*-celda es el *interior relativo* de un poliedro convexo de dimensión *k*. Las caras de una *k*-celda son los *interiores relativos* de las caras del poliedro correspondiente. Las caras de una *k*-celda son a su vez celdas de dimensión afín menor que *k*. □

El término *interior relativo* significa interior del conjunto relativo a la dimensión afín que tenga. En el caso en que la dimensión afín del conjunto sea nula (un punto) entenderemos que el interior relativo es el propio punto. Así, en el espacio E^2, el interior relativo de un triángulo es el triángulo menos sus lados, el de un segmento es el segmento menos sus vértices y el de un vértice es el propio vértice.

Definición 15: Esfera, circunsfera y esfera circunscrita.

Una esfera B de centro c y radio r es el conjunto de puntos de E^d cuya distancia a c es menor o igual a r. Una esfera B en un espacio de dimensión afín d está determinada por $d+1$ puntos afinmente independientes, $x_1,\cdots,x_{d+1}$. Si suponemos que las coordenadas del punto x_i son $(x_i^1,\cdots,x_i^d)$, entonces, las coordenadas $(x^1,\cdots,x^d)$ de un punto x de la superficie de la esfera asociada a los $d+1$ puntos están dadas por la ecuación:

$$\Delta(x,x_1,\cdots,x_{d+1})=\begin{vmatrix} l_1^2-l^2 & l_2^2-l_1^2 & \cdots & l_{d+1}^2-l_1^2 \\ x_1^1-x^1 & x_2^1-x_1^1 & \cdots & x_{d+1}^1-x_1^1 \\ \cdots & \cdots & \cdots & \cdots \\ x_1^d-x^d & x_2^d-x_1^d & \cdots & x_{d+1}^d-x_1^d \end{vmatrix}=0$$

donde $l^2=\sum_{m=1}^{d}(x^m)^2$ y $l_i^2=\sum_{m=1}^{d}(x_i^m)^2$, $1\le i\le d+1$.

Un punto x está dentro o sobre la superficie de B si y sólo si $\Delta(x,x_1,\cdots,x_{d+1})\le 0$.

Diremos que B es una circunsfera de un conjunto acotado S, si el interior relativo de S está contenido en B. En particular, si B es una circunsfera de un poliedro P y se cumple que $B \cap P$ son todos los vértices de P, entonces, B es la esfera circunscrita al poliedro P. □

Es bien sabido que sólo algunos poliedros admiten esferas circunscritas, entre ellos están los símplices.

Definición 16: Malla.

Una malla *conforme* de un poliedro $\Omega \subset E^d$ es un conjunto finito de poliedros $\{\Pi_i, 1 \le i \le n\}$ que verifica que:

1. $\Omega = \bigcup_{1 \le i \le n} \Pi_i$;

2. $\Pi_i \cap \Pi_j = \varnothing$ o una k-cara común a ambos poliedros, $1 \le i \ne j \le n$, $0 \le k \le d-1$. □

En esta definición la palabra *conforme* se emplea para indicar que no puede haber una intersección entre poliedros que no sea uno de sus vértices, aristas, etc.

Definición 17: Triangulación.

Una triangulación es una malla en la que todos los poliedros que la forman son símplices. □

Una triangulación en E^2 es una malla formada por triángulos, en E^3 está formada por tetraedros.

1.2. El diagrama de Voronoi y la triangulación de Delaunay.

El diagrama de Voronoi o teselación de Dirichlet asociado a un conjunto finito de puntos X es una partición del espacio euclideo E^d que asocia a cada punto de X la región del espacio más cercana. Lo más habitual es subdividir el plano mediante los llamados poliedros de Voronoi, $V(x_i)$. Estos poliedros se definen como el conjunto de puntos que distan lo mismo o menos de x_i que de cualquier otro punto de X, esto es:

$$V(x_i) = \{x \in E^d / \mathrm{d}(x, x_i) \leq \mathrm{d}(x, x_j), \forall x_j \in X, j \neq i\}$$

donde $\mathrm{d}(x,y)$ denota la distancia euclidea entre los puntos x e y. El diagrama de Voronoi sería el conjunto de todos estos poliedros. Esta definición tiene el inconveniente de que hay puntos del plano que pertenecen a varios poliedros a la vez, y por consiguiente, no establecen una verdadera partición del espacio. Si en la expresión anterior hubiéramos empleado una desigualdad estricta habría puntos que no pertenecerían a ninguno de estos conjuntos, y por tanto, tampoco se establecería una partición del espacio. Para evitar estos inconvenientes introduciremos el concepto de celda de Voronoi. Sea X un conjunto de n puntos distintos de E^d y sea R un subconjunto no vacío de X. Definimos la celda de Voronoi de R, $V(R)$, como el conjunto de los puntos de E^d que equidista de todos los puntos de R y es más cercano a un punto de R que a cualquier otro de $X - R$.

$$V(R) = \{x \in E^d / \mathrm{d}(x, x_i) = \mathrm{d}(x, x_j) \text{ y } \mathrm{d}(x, x_i) < \mathrm{d}(x, x_k), \forall x_i, x_j \in R, \forall x_k \in X - R\}$$

Si R está formado por un solo punto x_i, $V(R)$ es el conjunto de puntos de E^d que dista menos de x_i que de cualquier otro punto de X. La celda $V(R)$ puede ser un conjunto vacío, bien porque no halla ningún punto que equidiste de todos los de R, bien porque un punto que equidista de todos los de R también equidista de otro que está en $X - R$. En el caso en que el espacio sea E^3, las celdas de Voronoi asociadas a un solo punto son los interiores relativos de poliedros (celdas tridimensionales), las celdas asociadas a dos puntos son interiores relativos de polígonos (celdas bidimensionales), las asociadas a tres

puntos no colineales son interiores relativos de segmentos (celdas unidimensionales) y las asociadas a cuatro puntos no coplanarios son puntos del espacio (celdas de dimensión cero). En E^3, si R tiene cinco puntos o más que no están sobre una misma superficie esférica, $V(R)$ es un conjunto vacío. En general, en dimensión d, la frontera de una k-celda acotada ($k > 0$) está formada por l-celdas ($0 \leq l < k$). Las celdas no acotadas están asociadas a los puntos de X que determinan la envolvente convexa de X.

El diagrama de Voronoi $Vor(X)$ es el conjunto formado por todas las celdas no vacías $V(R)$, donde R recorre las partes del conjunto X.

$$Vor(X) = \{V(R) / R \in \text{Partes}(X) \text{ y } V(R) \neq \varnothing\}$$

El diagrama de Voronoi recubre todo el espacio, esto es, cualquiera que sea el punto de E^d existe un único conjunto R' tal que $V(R')$ contiene a dicho punto.

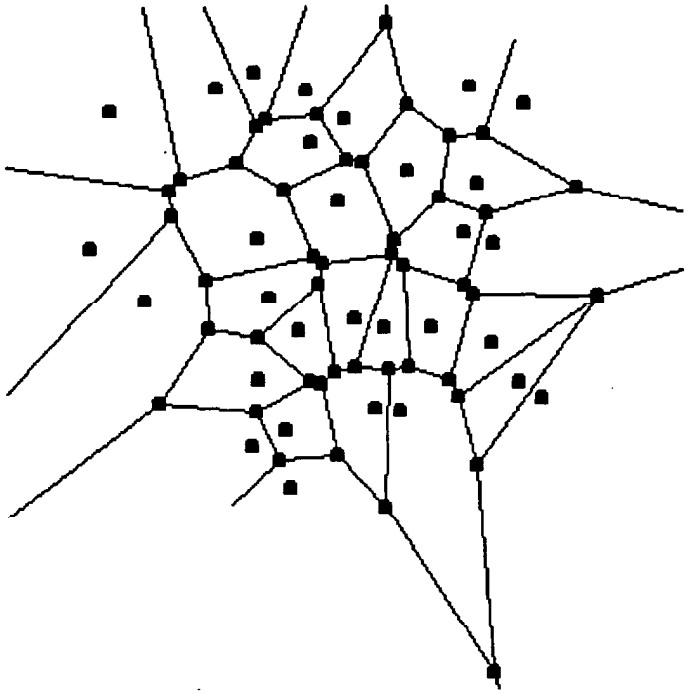


Figura 1. Diagrama de Voronoi de un conjunto de 30 puntos en disposición aleatoria.

Si R es un subconjunto de X tal que la celda de Voronoi $V(R)$ es no vacía, el interior de la envolvente convexa de R es una celda que se denomina celda de Delaunay y

se denota por $D(R)$. El conjunto de todas las celdas Delaunay, $Del(X)$, se denomina triangulación de Delaunay.

$$Del(X) = \{D(R) / R \in \text{Partes}(X) \text{ y } V(R) \neq \varnothing\}$$

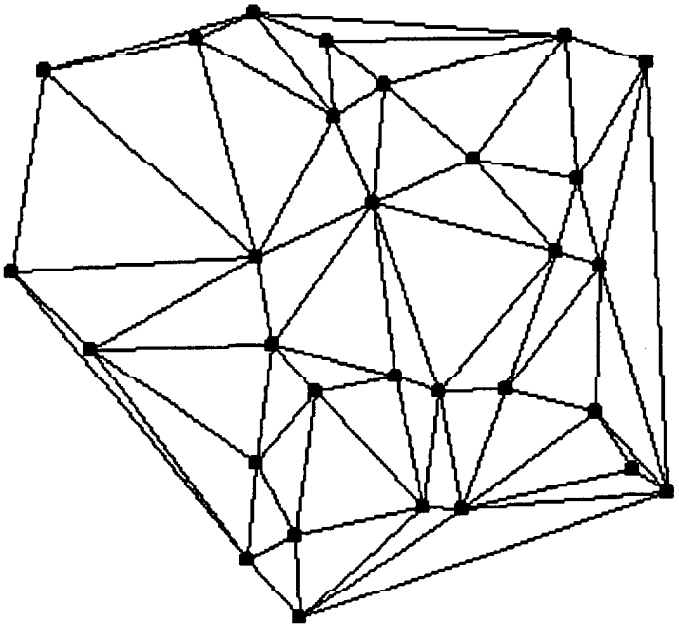


Figura 2. Triangulación de Delaunay del conjunto de puntos anterior.

Hay que hacer notar que, en sentido estricto, $Del(X)$ no es una triangulación ya que los elementos que constituyen este conjunto son celdas y no símplices. Ni siquiera la unión de las adherencias de las d-celdas forma, en general, una triangulación. La razón está en que, para disposiciones especiales de los puntos de X, algunas de las k-celdas $(0 \leq k \leq d)$ pueden estar determinadas por más de $k+1$ puntos de X, y en consecuencia, sus adherencias no son símplices.

Diremos que X está en *posición general* si y sólo si no hay más de $k+1$ puntos de X en la misma k-esfera $(2 \leq k \leq d)$. Cuando X está en posición general se verifica la condición $\text{card}(R) + \dim(V(R)) = d+1$. En esa situación, las k-celdas de Delaunay $D(R)$ estarán determinadas por $k+1$ puntos de X. En caso contrario habrá celdas $V(R)$ cuya

dimensión exceda $d+1-\text{card}(R)$; entonces diremos que $V(R)$ y $D(R)$ son *degeneradas* ($\text{card}(R)$ será mayor que $\dim(D(R))+1$).

Dado un conjunto de puntos, X, la triangulación de Delaunay, $Del(X)$, es única. Existe una correspondencia entre las celdas de Delaunay, $D(R)$, y las de Voronoi, $V(R)$. Una celda de Voronoi $V(R)$ de dimensión cero tiene asociada una celda de Delaunay $D(R)$ de dimensión d; si $V(R)$ tiene dimensión uno la celda $D(R)$ correspondiente tendrá dimensión d-1, y así sucesivamente de manera que la suma de las dimensiones de $V(R)$ y de $D(R)$ sea siempre d. Para entender esto hemos de darnos cuenta que, por ejemplo, una celda $V(R)$ de dimensión cero está formada como mínimo por d+1 puntos de X afinmente independientes, y por consiguiente, $D(R)$ tendrá dimensión d.

Para aclarar ideas, supongamos que X es un conjunto de puntos del plano euclideo E^2 y que $R=\{x_i,x_j,x_k\}$ es un subconjunto de X que determina una celda $V(R)$ de dimensión cero (el centro del círculo circunscrito al triángulo cuyos vértices son los puntos de R). La celda $D(R)$ será el interior de dicho triángulo (figura 3 (a)). De forma similar, partiendo de los conjuntos $R=\{x_i,x_j\}$ y $R=\{x_i\}$, se obtienen las celdas de Delaunay de dimensión uno y cero (figuras 3 (b) y (c)), respectivamente.

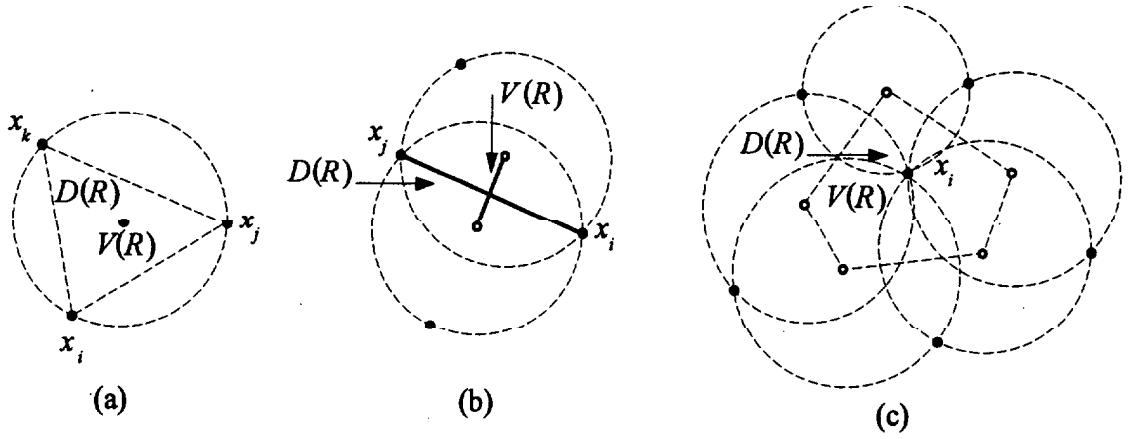


Figura 3. Celdas de Voronoi y Delaunay en el plano asociadas a los conjuntos $R=\{x_i,x_j,x_k\}$ (a), $R=\{x_i,x_j\}$ (b) y $R=\{x_i\}$ (c).

A continuación daremos algunas de las propiedades más importantes que satisfacen el diagrama de Voronoi y la triangulación de Delaunay. Las demostraciones se

pueden encontrar en [Fort92]. La única demostración que se expone aquí, por considerar que ayuda a comprender mejor el contenido de este trabajo, es la correspondiente a la propiedad de las esferas vacías. La importancia de esta propiedad es muy grande ya que caracteriza la triangulación de Delaunay. Además, algunos algoritmos de construcción de esta triangulación están basados en ella.

Supongamos que $Vor(X)$ y $Del(X)$ son el diagrama de Voronoi y la triangulación Delaunay, respectivamente, del conjunto X formado por n puntos de E^d.

Propiedad 1: Propiedad de las esferas vacías.

Si R es un subconjunto de X tal que existe $D(R)$ (R define una celda de Voronoi no vacía), entonces $D(R)$ admite una circunsfera que no contiene ningún punto de $X-R$. Recíprocamente, si R es un subconjunto de X tal que conv(R) admite una circunsfera que no contiene ningún punto de $X-R$, entonces $D(R)$ existe (coincide con el interior relativo de conv(R)).

Demostración:

Si $D(R)$ existe es porque R define una celda de Voronoi $V(R)$ no vacía. Cualquiera que sea el punto x de $V(R)$ debe equidistar de cualquiera de los x_i de R. Podemos construir una esfera centrada en x y que pase por los puntos x_i . Esta esfera no puede contener ningún punto x_k de $X-R$, ya que $d(x,x_i) < d(x,x_k)$ según la definición de celda de Voronoi.

Por otra parte, si R es un subconjunto de X tal que existe una circunsfera de conv(R) que no contiene ningún punto x_k de $X-R$, entonces el centro x de esta circunsfera equidista de los puntos x_i de R y además $d(x,x_i) < d(x,x_k)$. El punto x pertenecerá a $V(R)$ y por consiguiente existe la celda de Delaunay $D(R)$. □

Hemos de notar que esta propiedad, en el caso de dimensión tres, indica que cualquier celda de Delaunay, por ejemplo, el interior de un tetraedro o de una de sus caras etc., admite una circunsfera que no contiene ningún otro punto de X aparte de los

que determinan la celda. Y recíprocamente, si la envolvente convexa de un subconjunto $R \subseteq X$, por ejemplo un tetraedro, admite una circunsfera que no contiene ningún otro punto de X, la celda de Delaunay $D(R)$ (el interior del tetraedro) aparecerá en la triangulación de Delaunay.

Propiedad 2.

El diagrama de Voronoi, $Vor(X)$, es un *complejo de celdas* que establece una partición de E^d. □

Donde el significado de *complejo de celdas* es el de una colección de celdas disjuntas dos a dos y tales que sus caras son, a su vez, una celdas de la colección.

Propiedad 3.

La triangulación de Delaunay, $Del(X)$, es un complejo de celdas que establece una partición de la envolvente convexa de X. □

Propiedad 4.

El diagrama de Voronoi y la triangulación de Delaunay son duales, esto es, si R y R' son dos subconjuntos de X, una cara de $V(R)$ es una cara de $V(R')$ si y sólo si $D(R')$ es una cara de $D(R)$. □

Propiedad 5.

Si R es un subconjunto de X, entonces $V(R)$ es no acotada si y sólo si algún punto de R está en la frontera de la envolvente convexa de X. □

Las propiedades anteriores hacen referencia al diagrama de Voronoi y la triangulación de Delaunay en su conjunto. La propiedad que daremos a continuación caracteriza localmente la triangulación de Delaunay. Su importancia radica en que hay algunos algoritmos, como el *Flipping* o los de tipo incremental, como el de Watson [Wats81], que están basados en esta caracterización.

Definición 18.

Dos d-celdas opuestas (se entiende que dos d-celdas son opuestas si tienen una $(d-1)$-cara en común) celd(R) y celd(R') son *localmente de Delaunay* si admiten circunsferas, $B(R)$ y $B(R')$, tales que $R'-R \notin B(R)$ (o de forma equivalente, $R-R' \notin B(R')$). Una triangulación es *localmente de Delaunay* si cualquier par de d-celdas opuestas son de Delaunay . □

Propiedad 6.

Una triangulación de un conjunto de puntos X es de Delaunay si y sólo si es localmente de Delaunay. □

En este contexto, el significado de la palabra triangulación es el de complejo de celdas con vértices en X y cuya unión es la envolvente convexa de X.

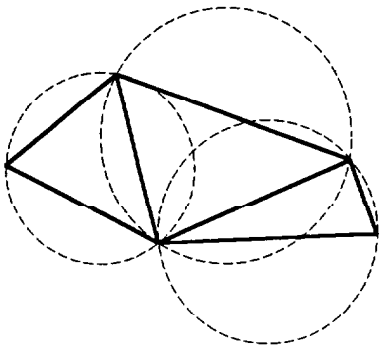


Figura 4. Triangulación localmente de Delaunay.

La triangulación de la figura 4 es localmente de Delaunay ya que todas las d-celdas opuestas son de Delaunay (sólo se muestran las esferas circunscritas a las 2-celdas). En consecuencia, dicha triangulación es de Delaunay.

La definición de la triangulación de Delaunay como un complejo de celdas resulta útil para establecer sus propiedades. No obstante, la nomenclatura empleada por la mayor parte de los artículos y textos utiliza una definición similar a la dada en la sección

dedicada a los preliminares. Para evitar confusión definiremos la triangulación de Delaunay (en el sentido de la definición 17), de un conjunto de puntos, X, en posición general, como:

$$\mathrm{D}(X) = \bigcup \overline{D(R)} \,/\, D(R) \in Del(X) \text{ y } \dim(D(R)) = d$$

Cuando X no esta en posición general habrá celdas cuyas adherencias no sean símplices. En ese caso $\mathrm{D}(X)$ es una malla, pero no una triangulación. A pesar de ello, siempre será posible descomponer en símplices aquellos politopos que no lo sean y con ello transformar la malla en una triangulación. Ésta es una operación delicada desde el punto de vista computacional ya que la descomposición en símplices de un politopo correspondiente a una celda degenerada no es única. De hecho, los algoritmos que realizan la triangulación de Delaunay encuentran dificultades cuando X no está en posición general. Las dificultades subsisten incluso cuando la disposición de puntos es tal que X no está *claramente* en posición general, es decir, cuando hay más de $k+1$ puntos de X *próximos* a la misma k-esfera. Estudiaremos estos problemas en detalle en el apartado dedicado a errores de redondeo.

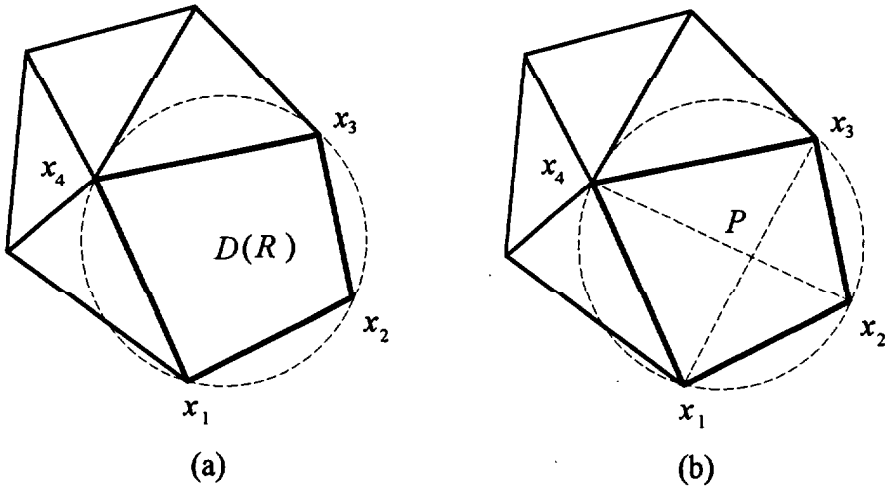


Figura 5. Triangulación de Delaunay en el plano con 2-celda $D(R)$ degenerada (a). *Dos posibles descomposiciones en símplices del politopo correspondiente a $D(R)$* (b).

La triangulación de Delaunay presenta algunas propiedades que la hacen muy adecuada para la aplicación del MEF. En concreto, para un determinado conjunto de puntos del plano es la triangulación que produce los triángulos más regulares.

Propiedad 7.

De todas las triangulaciones de un conjunto de puntos $X \subset E^2$ en posición general, la de Delaunay es aquella que hace máximo el ángulo mínimo de cualquier triángulo. □

En dimensión mayor que dos no existe una generalización de la propiedad anterior, al menos en términos de alguna medida angular de los símplices. Un criterio que sí es aplicable a cualquier dimensión es el de la *circunsfera mínima*. La circunsfera mínima de un símplice K es la menor de todas las esferas que contienen a K. Si el centro de la esfera circunscrita a un símplice está dentro del símplice, entonces la circunsfera mínima es la esfera circunscrita. En caso contrario, el centro de la circunsfera mínima está sobre alguna de las caras del símplice.

Propiedad 8.

Supongamos que T(X) es una triangulación cualquiera de un conjunto de puntos $X \subset E^d$ en posición general y maxrad(T) es el máximo radio de cualquiera de las esferas mínimas. El mínimo valor de maxrad(T) se obtiene cuando T(X) es la triangulación de Delaunay de X. □

En realidad esta propiedad no nos da una buena idea de la regularidad de la triangulación de Delaunay. De hecho, es un problema abierto el encontrar alguna otra medida de calidad que la triangulación de Delaunay haga óptima. [BerEpp92].

1.3. Algoritmos de construcción de la triangulación de Delaunay. Generalidades.

La complejidad de un algoritmo se mide por el número de operaciones elementales que debe realizar para concluir un proceso. Para aclarar qué se entiende por operación elemental hay que precisar el tipo de máquina con la que se trabaja. En el contexto de la geometría computacional se hace la abstracción de suponer que las operaciones se efectúan en una *máquina de acceso aleatorio* (RAM). Esta máquina es capaz de almacenar un número *real* en cada posición de memoria y realizar operaciones aritméticas $(+,-,\times,\div)$ o comparaciones entre números reales $(<,\leq,=,\neq,\geq,>)$ con un coste unitario de tiempo [PreSha85]. Con estas suposiciones está claro que el tiempo de ejecución de un algoritmo es proporcional a su complejidad.

El número exacto de operaciones es extremadamente difícil, si no imposible, de predecir; además, no tiene especial relevancia. Por este motivo, la complejidad de un algoritmo se estudia estableciendo una cota superior $O(f(n))$ y una cota inferior $\Omega(f(n))$. El término $O(f(n))$ denota el conjunto de funciones $g(n)$ tales que existen unas constantes positivas C y n_0 de manera que $g(n) \leq Cf(n)\ \ \forall n \geq n_0$. Por su parte, el término $\Omega(f(n))$ denota las funciones tales que $g(n) \geq Cf(n)\ \ \forall n \geq n_0$ [PreSha85].

La cota inferior para la complejidad de los algoritmos que realizan la triangulación de Delaunay de n puntos en dimensión dos está en $\Omega(n \log n)$, mientras que para dimensión d mayor o igual a tres es $\Omega(n^{\lceil d/2 \rceil})$.

En dimensión dos existen gran número de algoritmos, algunos de ellos específicos de esta dimensión. Por ejemplo, el algoritmo conocido como *Flipping* parte de una triangulación de un conjunto de n puntos del plano y analiza si cada par de triángulos opuestos son localmente de Delaunay. En caso de que no sea así, intercambia la diagonal consiguiendo de esta manera que el par de triángulos sea localmente de Delaunay. Se puede demostrar que este proceso es finito y que su complejidad es $\Omega(n^2)$ [Fort92].

En dimensión arbitraria d hay una importante clase de algoritmos basados en la

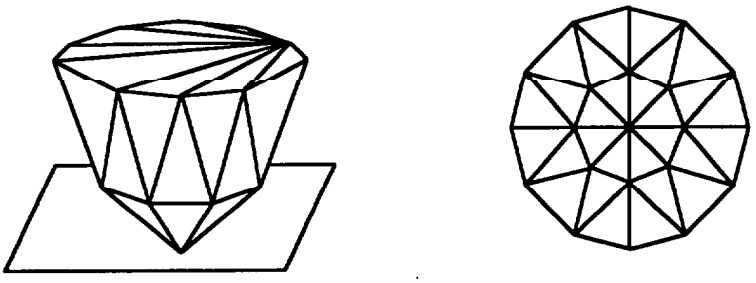
creación de la envolvente convexa de las proyecciones de los puntos sobre un paraboloide en dimensión $d+1$. Para hacer que sea más sencillo imaginar cómo funcionan estos algoritmos supondremos que X es un conjunto de puntos en dimensión dos. Se define la aplicación $\lambda: E^2 \rightarrow E^3$ mediante la relación

$$\lambda(x) = (\xi_1, \xi_2, \xi_1^2 + \xi_2^2)$$

donde (ξ_1, ξ_2) son las coordenadas del punto $x \in E^2$. La imagen de esta aplicación es el paraboloide dado por:

$$\Lambda = \{(x,\zeta) \in E^3 / x \in E^2,\ \zeta = \xi_1^2 + \xi_2^2\}$$

Se puede demostrar [Fort92] que la triangulación de Delaunay $Del(X)$ es la proyección sobre el plano de la base de las caras *inferiores* de la envolvente convexa de $\lambda(X)$. Se entiende por caras *inferiores* aquellas que son *visibles* desde el punto del eje ζ $(0,0,-\infty)$. Una cara es *visible* desde un punto p si cualquiera que sea el punto q de la cara el segmento pq no intersecta ninguna región *prohibida*, en nuestro caso la envolvente convexa de $\lambda(X)$.



(a) (b)

Figura 6. Envolvente convexa de $\lambda(X)$ (a) *y proyección sobre el plano de la base* (b).

Existen numerosos algoritmos para hallar la envolvente convexa de un conjunto de puntos en un espacio E^d que pueden ser utilizados a su vez para construir la triangulación de Delaunay. Sin entrar en detalles, explicaremos a continuación las principales ideas de uno de estos algoritmos.

En primer lugar se elige un subconjunto R de X con $d+2$ puntos de manera que $\lambda(R)$ determine un símplice de E^{d+1}. Obviamente la envolvente convexa H de $\lambda(R)$ coincide con este símplice. Tomando como punto de partida H construiremos la envolvente convexa de X añadiendo puntos uno a uno y modificando oportunamente la envolvente convexa precedente.

Supongamos ahora que ya ha sido construida la envolvente convexa H de un subconjunto $R \subseteq X$. La modificación de H cuando se añade el nuevo punto x_{i+1} se lleva a cabo como sigue:

1. Encontramos el conjunto H_v de d-caras visibles desde $\lambda(x_{i+1})$.
2. Substituimos H_v por el conjunto de d-caras definido por:

$$H_s = \{F_s / F_s = \text{cell}(F \cup \{\lambda(x_{i+1})\}), F \in H_h\}$$

donde H_h es el conjunto de $(d-1)$-caras que definen el *horizonte* de H *visto* desde $\lambda(x_{i+1})$ y $\text{cell}(F \cup \{\lambda(x_{i+1})\})$ es la celda formada a partir de $F \in H_h$ y el punto $\lambda(x_{i+1})$. □

El proceso anterior se repite hasta que no quedan más puntos de X por incluir; es por tanto un algoritmo incremental. La triangulación de Delaunay se obtiene proyectando las caras *inferiores* de H sobre el plano $\zeta = 0$.

La complejidad del algoritmo depende de lo rápidamente que encontremos una de las d-caras de H_v. Supongamos que los puntos de X se ordenan según su coordenada ξ_1, y en caso de haber varios con la misma coordenada ξ_1, según su coordenada ξ_2. Si los puntos se añaden en este orden, cada nuevo punto $\lambda(x_{i+1})$ siempre *ve* una de las d-caras, digamos F, de H_v. La búsqueda del resto de d-caras de H_v se efectúa partiendo de F y continuando por las d-caras *vecinas* de las ya incluidas en H_v. Es posible demostrar que la complejidad de este algoritmo, cuando los puntos se añaden en el orden referido, es $O(n^{\lceil d+1/2 \rceil})$.

Otro tipo de algoritmo incremental muy utilizado es el conocido como algoritmo de Watson [Wats81]. La ventaja que presenta éste frente a otros algoritmos incrementales es su simplicidad conceptual. Éste es el motivo que nos ha hecho decidirnos por este algoritmo, o para ser exactos por una variante del mismo, para construir nuestro generador de mallas. Además, como el resto de algoritmos incrementales, permite refinar localmente la malla en las zonas donde el error cometido por el MEF es mayor. El refinamiento de la malla se puede acometer sin necesidad de reconstruirla desde el principio, bastará con introducir nuevos puntos, en posiciones adecuadas, en la malla existente. En el capítulo siguiente haremos una exposición detallada de esta variante del algoritmo de Watson, de las dificultades que conlleva su *implementación* en el ordenador y de las modificaciones que hemos realizado para salvar estas dificultades.

2. ALGORITMO DE TRIANGULACIÓN

2.1. Introducción.

El algoritmo que vamos a presentar aquí es una variante del conocido algoritmo de Watson [Wats81]. Esta versión está reflejada en el articulo de P. L. George titulado *"Delaunay's mesh of a convex polyhedron in dimension d. Application to arbitrary polyhedra"* [GeoHec92]. La realización de este algoritmo presenta importantes problemas, debidos a errores de redondeo, cuando los puntos no están *claramente* en posición general. La solución que propone George en su articulo hace uso de aritmética exacta en algunos cálculos. Ésto supone, como veremos en el apartado 2.3, la aplicación de un factor de escala a los puntos de X, y que éstos, una vez aplicado el factor de escala, deban estar situados sobre los nodos de una *rejilla* de $511 \times 511 \times 511$ (en dimensión tres).

Nosotros nos proponemos acometer estos problemas sin hacer uso de aritmética exacta, esto es, utilizando en todo momento *números en coma flotante*. La ventaja de hacerlo así está en el ahorro de tiempo de CPU y en una menor restricción en la ubicación de los puntos.

También dentro de este capítulo, se exponen algunas de las estructuras de datos capaces de soportar la información de una triangulación. La elección de una u otra depende en gran medida del uso posterior de la triangulación. La escogida por nosotros pretende ser lo más versátil posible, aún a costa de mayor consumo de memoria.

Al final del capítulo se dan los detalles de cómo el algoritmo actualiza los datos de la triangulación cada vez que se añade un nuevo punto.

2.2. Descripción del algoritmo.

Consideremos el conjunto finito de puntos, no todos coplanarios, $X \subset E^d$. El algoritmo que expondremos a continuación parte de la triangulación de los $d+1$ primeros puntos, supuestos afinmente independientes, y prosigue añadiendo puntos de X, haciendo las oportunas modificaciones de la triangulación, hasta que se llega al último punto. Los principales pasos que efectúa el algoritmo son los siguientes:

Construcción de la triangulación inicial.

Supongamos que los $d+1$ primeros puntos, X_{d+1}, de X son afinmente independientes. El símplice K_1 determinado por los puntos de X_{d+1} constituye una triangulación de Delaunay de X_{d+1}.

Análisis de la posición del punto x_{i+1} respecto de la triangulación.

Supondremos que ya ha sido construida la triangulación $\mathrm{D}(X_i)$ de los i primeros puntos de X. Designaremos por K_j a los símplices que forman la triangulación y por $B(K_j)$ a sus esferas circunscritas. Supondremos que estos símplices están orientados *correctamente*, es decir, $\det(K_j) > 0$. Antes de añadir el nuevo punto, x_{i+1}, debemos conocer su posición respecto de $\mathrm{D}(X_i)$.

Sea $\mathrm{B}(X_i) = \bigcup_{1 \le j \le i} B(K_j)$ la cobertura de las esferas circunscritas a los símplices de $\mathrm{D}(X_i)$. Está claro que sólo una de las tres situaciones siguientes es posible:

(a) $x_{i+1} \in \mathrm{D}(X_i)$, esto es, $\exists\, K_j \in \mathrm{D}(X_i) / x_{i+1} \in K_j$;

(b) $x_{i+1} \notin \mathrm{B}(X_i)$, esto es, $x_{i+1} \notin B(K_j), 1 \le j \le i$;

(c) $x_{i+1} \notin \mathrm{D}(X_i)$ pero $x_{i+1} \in \mathrm{B}(X_i)$, esto es, $\exists\, K_j \in \mathrm{D}(X_i) / x_{i+1} \in B(K_j)$. □

Para caracterizar estas situaciones utilizaremos unos símplices *virtuales* K_j^*, $1 \le j \le p$, construidos con las caras de la frontera de $D(X_i)$ y el punto x_{i+1}. Las caras de la frontera de $D(X_i)$ son aquellas que pertenecen a un único símplice de $D(X_i)$; son caras de la envolvente convexa de X_i.

(1) La situación (a) se caracteriza por:

$$\det(K_j^*) \ge 0 \quad \forall j, 1 \le j \le p$$

(2) La situación (b) está caracterizada por:

$$\Delta(x_{i+1}, K_j) > 0 \quad \forall K_j \in D(X_i)$$

donde $\Delta(x_{i+1}, K_j)$ es el determinante asociado a la esfera circunscrita al símplice K_j (ver definición 13).

(3) Por último, vemos que la situación (c) es la negación de las dos anteriores

$$\exists j, 1 \le j \le p / \det(K_j^*) < 0 \quad \text{y} \quad \exists K_j \in D(X_i) / \Delta(x_{i+1}, K_j) \le 0. \square$$

Inclusión del punto x_{i+1} en la triangulación.

La triangulación $D(X_i)$ debe modificarse parcialmente para dar lugar a la triangulación $D(X_{i+1})$ que incluya al punto x_{i+1}. Según la posición en la que se encuentre x_{i+1} respecto de la triangulación se procederá de una de las siguientes maneras:

(1) Situación (a) (ver figura 7): Hallamos el conjunto de símplices cuyas esferas circunscritas contienen a x_{i+1}

$$D_1^i = \{K_j \in D(X_i) / x_{i+1} \in B(K_j)\}$$

y determinamos las caras de su frontera, $C_{fD_1}^i$. La triangulación de los $i+1$ primeros puntos es:

$$D(X_{i+1}) = (D(X_i) - D_1^i) \cup D_2^i$$

donde $D_2^i = \bigcup_j \text{conv}(C_j \cup \{x_{i+1}\})$ es el conjunto de símplices formados con las cada cara $C_j \in C_{fD_1}^i$ y el punto x_{i+1}. El hecho de que las esferas circunscritas a los símplices de D_1^i contengan el punto x_{i+1} asegura que las caras de $C_{fD_1}^i$ sean visibles (una vez eliminados los tetraedros de D_1^i) desde dicho punto; esto asegura la consistencia topológica de la construcción. En otras palabras, no se producirán *cruces* entre los símplices recién formados y los ya existentes.

(2) Situación (b) (ver figura 8): Encontramos las caras de $D(X_i)$ que son visibles desde x_{i+1} (estas caras serán forzosamente de la frontera de $D(X_i)$). Al conjunto de todas ellas lo denominaremos C_{vD}^i. En este caso, la triangulación $D(X_{i+1})$ está dada por:

$$D(X_{i+1}) = D(X_i) \bigcup_j \text{conv}(C_j \cup \{x_{i+1}\}) \quad \forall C_j \in C_{vD}^i$$

(3) Situación (c) (ver figura 9): Sea $R^i = D(X_i) - D_1^i$ el conjunto resultante de restar a $D(X_i)$ el conjunto de símplices D_1^i. Denominemos C_{vR}^i a las caras de R^i visibles desde x_{i+1}. En este caso, la triangulación $D(X_{i+1})$ está dada por:

$$D(X_{i+1}) = R^i \bigcup_j \text{conv}(C_j \cup \{x_{i+1}\}) \quad \forall C_j \in C_{vR}^i. \square$$

En todos los casos anteriores se supone que la orientación de los símplices es tal que sus determinantes asociados son positivos. El proceso empleado para construir la triangulación de los $i+1$ puntos asegura que ésta sea localmente de Delaunay, y en consecuencia, según la propiedad 6, la triangulación final será de Delaunay. La triangulación concluye cuando se han añadido todos los puntos de X.

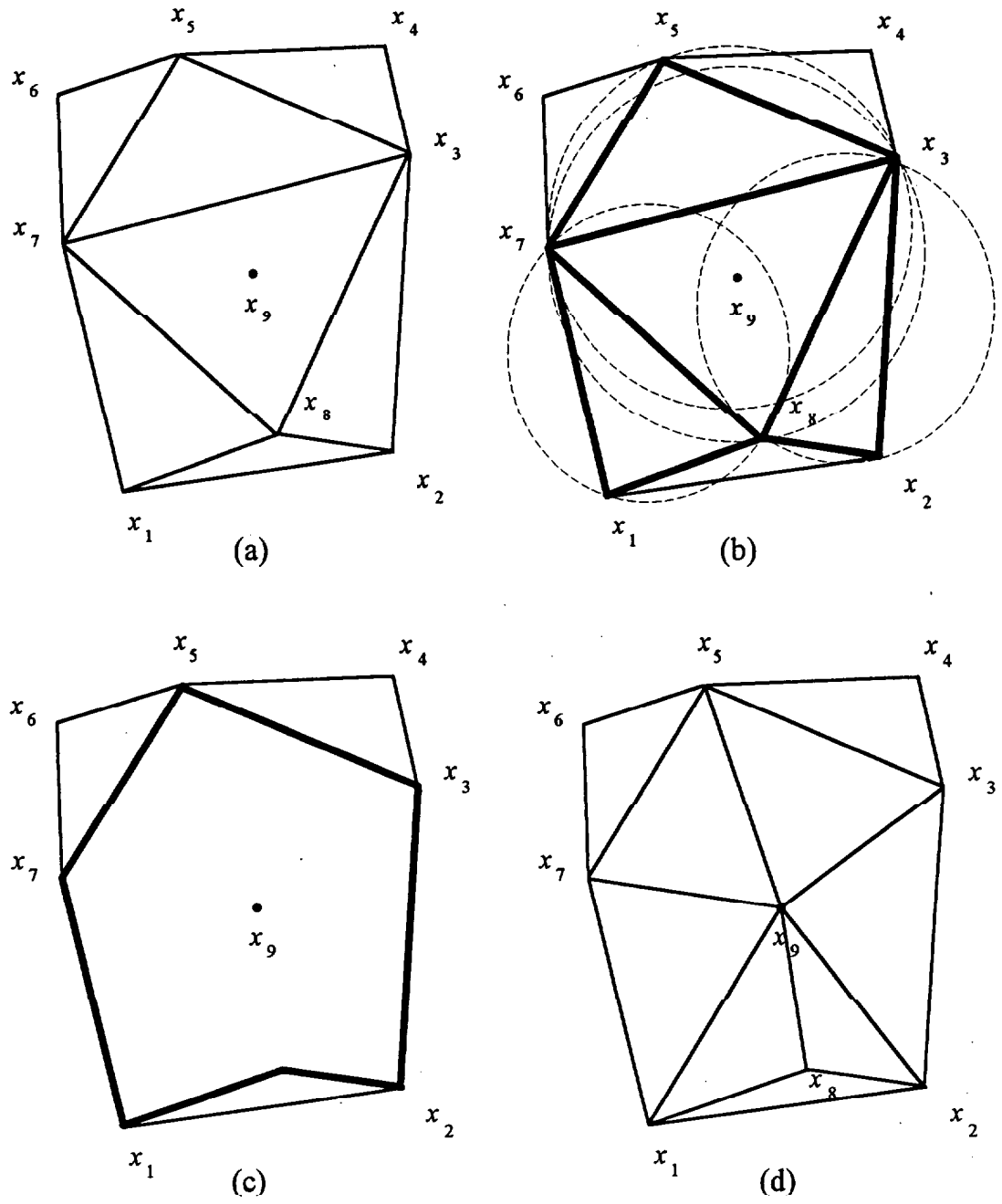


Figura 7. (*Situación* (a)). *Inclusión del punto* x_9 *en la triangulación* $\mathrm{D}(X_8)$ (a). *Conjunto* D_1^8 *(líneas gruesas)* (b). *Frontera de* D_1^8 *(líneas gruesas)* (c). *Triangulación final,* $\mathrm{D}(X_9)$.

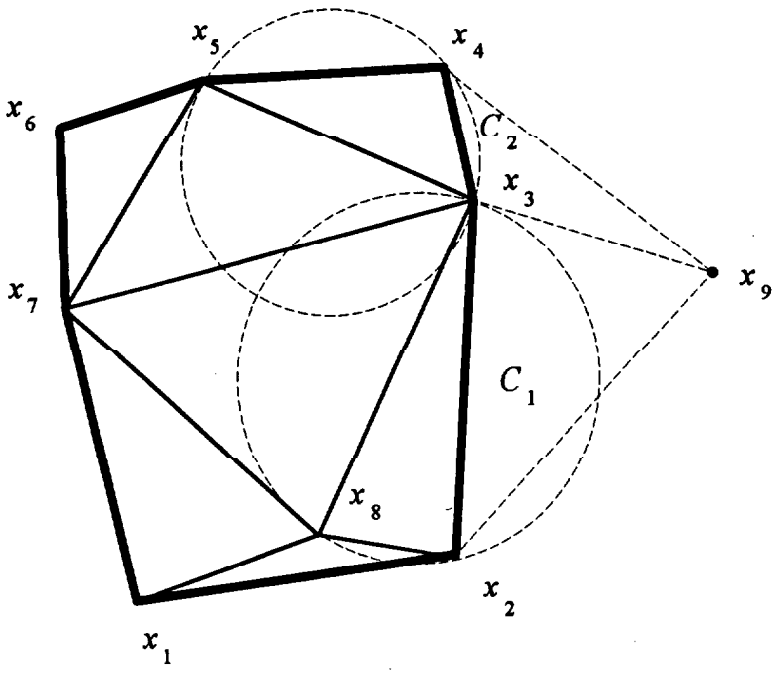


Figura 8. (*Situación* (b)). *Inclusión del punto* x_9 *en la triangulación (Sólo se muestran los círculos más próximos a* x_9*). Las caras visibles desde* x_9 *son* $C^8_{vD} = \{C_1, C_2\}$.

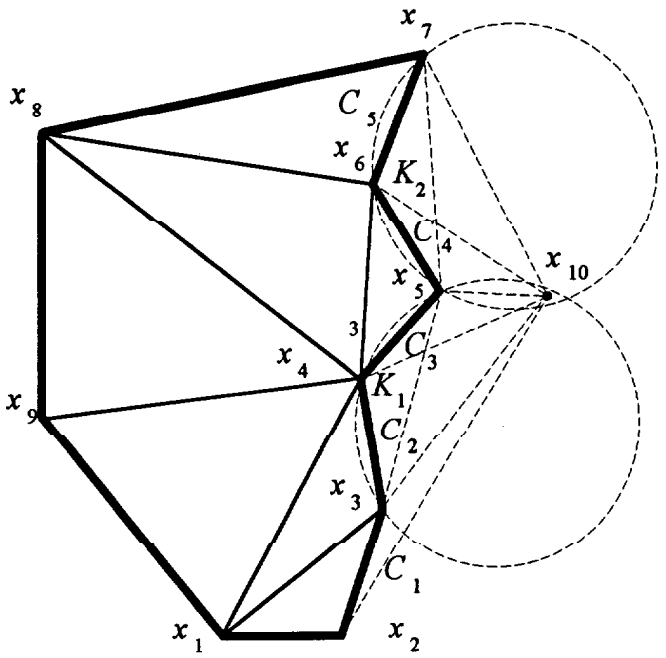


Figura 9. (*Situación* (c)). *Inclusión del punto* x_{10} *en la triangulación (Sólo se muestran los círculos que contienen a* x_{10}*). El conjunto de caras visibles desde* x_{10}*, después de eliminar los triángulos* K_1 *y* K_2*, es* $C^9_{vR} = \{C_1, \cdots, C_5\}$.

El algoritmo mostrado anteriormente construye la envolvente convexa de X. Es posible simplificarlo notablemente si hacemos que todos los puntos de X estén incluidos en un símplice inicial, K_0, o en cualquier otro d-politopo descompuesto en símplices formando una triangulación de Delaunay. Está claro que de las tres situaciones anteriores, la única que se puede presentar ahora es la situación (a). Con esta simplificación vemos que la eficacia del algoritmo depende fundamentalmente del procedimiento empleado para encontrar los elementos de D_1^i; los procesos restantes se circunscriben a la región determinada por D_1^i.

2.3. Problemas debidos a errores de redondeo.

En teoría, el algoritmo descrito anteriormente es capaz de construir la triangulación de Delaunay de cualquier conjunto de puntos, incluso cuando éstos no están en posición general. El que ésto sea así depende de que los cálculos que determinan la situación del punto x_{i+1} respecto de la triangulación, la orientación de los símplices y el conjunto de símplices que desaparecen, D_1^i, se haga de forma exacta. La situación del punto x_{i+1} está dada por los signos de los determinantes $\det(K_j^*)$ y $\Delta(x_{i+1},K_j)$, la orientación de un símplice depende del signo de $\det(K_j)$ y el que un símplice pertenezca o no a D_1^i está determinada por el signo de $\Delta(x_{i+1},K_j)$; $x_{i+1} \in B(K_j)$ si y sólo si $\Delta(x_{i+1},K_j) \leq 0$. La única manera de evaluar los signos de estos determinantes es calculándolos. En ocasiones el valor absoluto de estos determinantes es tan pequeño que puede sobrepasar la tolerancia del ordenador cuando éste trabaja con *números en coma flotante*. Una evaluación errónea en el signo de un determinante puede hacer, como veremos más adelante, que la construcción que realiza el algoritmo no sea, desde un punto de vista topológico, una triangulación.

Una alternativa al cálculo con *números en coma flotante* es el empleo de aritmética exacta. Los problemas de memoria y tiempo de CPU que esto conlleva serán examinados también dentro de este apartado.

En el análisis que haremos a continuación de las diversas situaciones que dan lugar a triangulaciones topológicamente incorrectas, o por lo menos *conflictivas*, supondremos que la precisión con la que trabaja el ordenador es limitada, esto es, se emplea aritmética no exacta.

Una situación que se puede presentar en cualquier dimensión, y que por simplificar describiremos en dimensión dos, ocurre cuando los puntos de X no están *claramente* en posición general. Supongamos que $X = \{x_1,\cdots,x_5\}$ está formado por

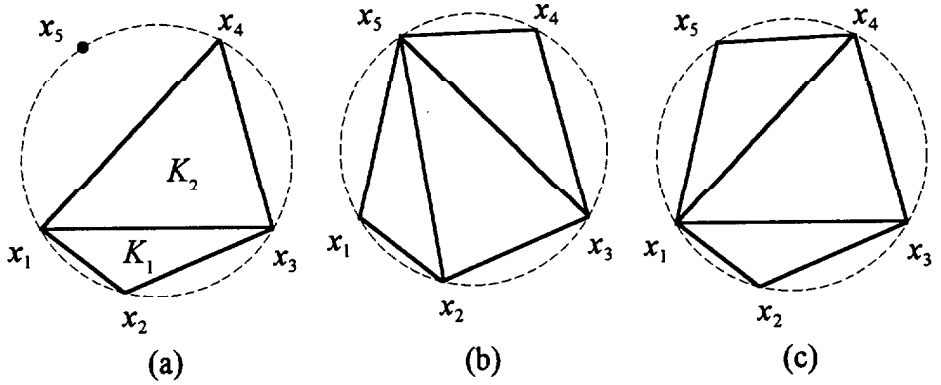
cinco puntos de E^2 situados sobre la misma circunferencia. Supongamos también que ya se ha realizado la triangulación de los cuatro primeros puntos dando lugar a los triángulos $K_1 = \{x_1, x_2, x_3\}$ y $K_2 = \{x_1, x_3, x_4\}$ (figura 10 (a)). La imprecisión en la evaluación de los determinantes $\Delta(x_5, K_1)$ y $\Delta(x_5, K_2)$, cuando se va a incluir el punto x_5 en la triangulación, puede dar lugar a las cuatro situaciones siguientes:

(a) Si el ordenador evalúa que $\Delta(x_5, K_1) \leq 0$ y $\Delta(x_5, K_2) \leq 0$, los triángulos K_1 y K_2 desaparecen dando lugar a los nuevos triángulos $\{x_1, x_2, x_5\}$, $\{x_2, x_3, x_5\}$ y $\{x_3, x_4, x_5\}$ (figura 10 (b)).

(b) Si evalúa que $\Delta(x_5, K_1) > 0$ y $\Delta(x_5, K_2) > 0$, los triángulos K_1 y K_2 permanecen y se forma el nuevo triángulo $\{x_1, x_4, x_5\}$ (figura 10 (c)).

(c) Si evalúa que $\Delta(x_5, K_1) > 0$ y $\Delta(x_5, K_2) \leq 0$, el triángulo K_1 permanece, el K_2 desaparece y se forman los nuevos símplices $\{x_1, x_3, x_5\}$ y $\{x_3, x_4, x_5\}$ (figura 10 (d)).

(d) Por último, si evalúa que $\Delta(x_5, K_1) \leq 0$ y $\Delta(x_5, K_2) > 0$, el triángulo K_1 desaparece, el K_2 permanece y se forman los nuevos triángulos $\{x_1, x_2, x_5\}$ y $\{x_2, x_3, x_5\}$ (figura 10 (e)).



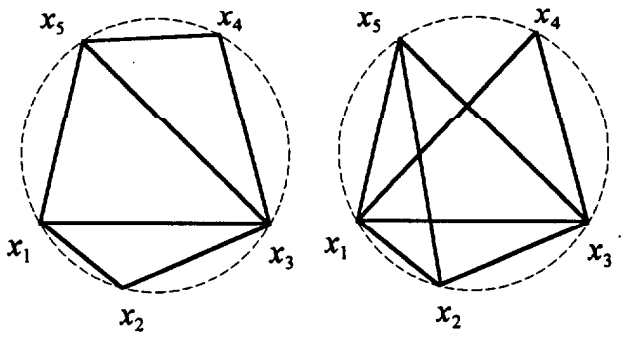


Figura 10. Diversas posibilidades, debidas a errores de redondeo, en la triangulación de cinco puntos coesféricos.

Todas las situaciones anteriores son admisibles a excepción de la última ya que en ella se produce un *cruce* entre triángulos. La topología de esa construcción no corresponde a una triangulación. En la práctica, la situación anterior se presenta con frecuencia si no se arbitra algún procedimiento que la evite, incluso cuando los puntos no están sobre la misma circunferencia pero están próximos a ella.

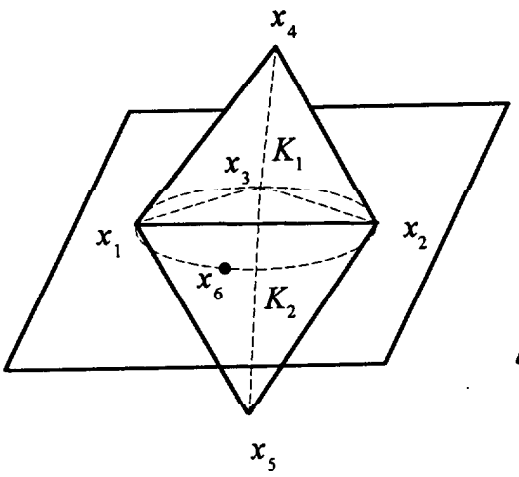
Otra situación que puede dar lugar a construcciones incorrectas, y ésta sólo ocurre en dimensión mayor o igual que tres, se produce cuando se incluye un nuevo punto sobre la frontera de la $(d-1)$-esfera circunscrita a la cara común de dos símplices opuestos. Como veremos más adelante, puede ocurrir que aparezcan *símplices* degenerados con volumen nulo, e incluso *cruces* entre ellos. Para que la comprensión de esta circunstancia resulte más sencilla, analizaremos la situación suponiendo que la triangulación es tridimensional. Consideremos que el conjunto $X=\{x_1,\cdots,x_6\}$ está formado por seis puntos de E^3 y que cuatro de ellos, x_1, x_2, x_3 y x_6, están situados sobre la misma circunferencia. Si la triangulación de los cinco primeros ha dado lugar a los tetraedros $K_1=\{x_1,x_2,x_3,x_4\}$ y $K_1=\{x_1,x_2,x_3,x_5\}$ (ver figura 11 (a)), la inclusión del punto x_6 en la triangulación puede dar lugar a las cuatro situaciones siguientes:

(a) Si el ordenador evalúa que $\Delta(x_6,K_1)\leq 0$ y $\Delta(x_6,K_2)\leq 0$, los tetraedros K_1 y K_2 desaparecen y se forman los nuevos tetraedros $\{x_1,x_3,x_4,x_6\}$, $\{x_2,x_3,x_4,x_6\}$, $\{x_1,x_3,x_5,x_6\}$ y $\{x_2,x_3,x_5,x_6\}$ (figura 11 (b)).

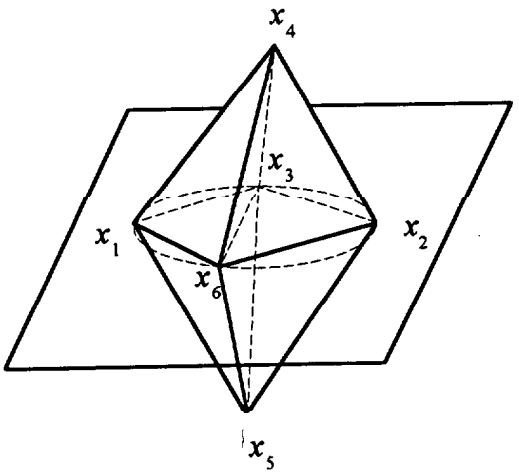
(b) Si evalúa que $\Delta(x_6,K_1)>0$ y $\Delta(x_6,K_2)>0$, los tetraedros K_1 y K_2 permanecen y se forman los nuevos tetraedros $\{x_1,x_2,x_4,x_6\}$ y $\{x_1,x_2,x_5,x_6\}$ (figura 11 (c)).

(c) Si evalúa que $\Delta(x_6,K_1)\leq 0$ y $\Delta(x_6,K_2)>0$, el tetraedro K_1 desaparece, el K_2 permanece y se forman los nuevos tetraedros $\{x_1,x_2,x_3,x_6\}$, $\{x_1,x_3,x_4,x_6\}$, $\{x_2,x_3,x_4,x_6\}$ y $\{x_1,x_2,x_5,x_6\}$ (figura 11 (d)).

(d) Por último, si evalúa que $\Delta(x_6,K_1)>0$ y $\Delta(x_6,K_2)\leq 0$, el tetraedro K_1 permanece, el K_2 desaparece y se forman los nuevos tetraedros $\{x_1,x_2,x_3,x_6\}$, $\{x_1,x_3,x_5,x_6\}$, $\{x_2,x_3,x_5,x_6\}$ y $\{x_1,x_2,x_4,x_6\}$ (figura 11(d)).



(a)



(b)

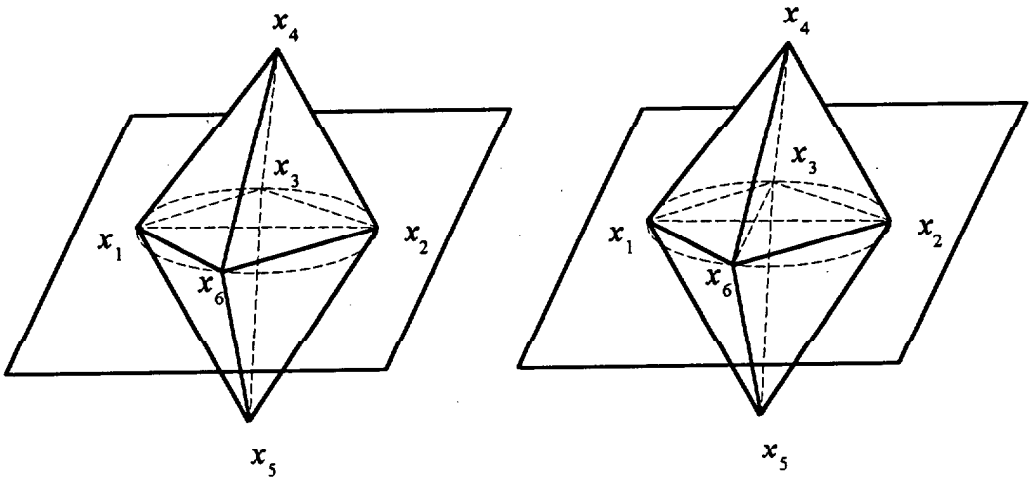


Figura 11. Diferentes posibilidades en la triangulación del conjunto de puntos $X = \{x_1, \cdots, x_6\}$.

Las dos primeras construcciones (figuras 11 (b) y 11 (c)) corresponden a triangulaciones admisibles. Las dos últimas están representadas por la figura 11 (d). En ellas aparecen *tetraedros* degenerados, es decir *tetraedros* planos con volumen nulo. Son, por consiguiente, construcciones inadmisibles en la triangulación. Hay que hacer notar que, incluso trabajando con aritmética exacta, se puede llegar a situaciones en la que aparezcan tetraedros arbitrariamente planos y por tanto no aptos para la aplicación del MEF. Bastaría para ello que el punto x_6 estuviera situado, por ejemplo, en la superficie de la esfera circunscrita a K_1 y por encima del plano determinado por los puntos x_1, x_2 y x_3 a una distancia de éste suficientemente pequeña. Desde el punto de vista del MEF son preferibles las triangulaciones de las figuras 11 (b) y 11 (c) aunque éstas no fueran, estrictamente, de Delaunay.

En caso de operar con aritmética no exacta podría ocurrir, suponiendo que la situación real del punto x_6 es la misma de antes, que el ordenador interpretara que $x_6 \notin B(K_1)$ y $x_6 \in B(K_2)$. Esto conduciría a una construcción topológicamente incorrecta en la que aparecerían *cruces* entre tetraedros.

Se han propuesto numerosas formas de solucionar estos problemas. Una de ellas consiste en aplicar un factor de escala a los datos de la entrada de manera que los números en coma flotante se conviertan en enteros. Dado que los determinantes de números enteros son a su vez números enteros, es posible utilizar aritmética exacta para evaluarlos sin error alguno. Uno de los problemas que plantea la utilización de aritmética exacta es la limitación en el valor de las cantidades que aparecen en los determinantes. Supongamos que b sea el número de bytes de la mantisa de una palabra y L la longitud típica del dominio después de haber aplicado el factor de escala. El determinante $\Delta(x_{i+1}, K_j)$ incluye términos tales como $\sum_{i=1}^{d+1} L_i^{d+2}$, donde L_i son cantidades similares a L. Este determinante puede ser computado con enteros si se cumple que $L_i \leq 2^{b/d+2} - 1$. Para $d = 3$ y $b = 48$, resulta $L_i = 2^9 - 1 = 511$. Esto significa que todos los puntos de X, una vez se han normalizado, deben encontrarse en una *rejilla* del orden de $511 \times 511 \times 511$. Esta limitación es claramente excesiva para muchas aplicaciones de interés práctico. Otro factor que hay que tener en cuenta es el substancial aumento de tiempo de CPU que conllevan los cálculos con aritmética exacta.

Una alternativa al uso de aritmética exacta en todos los cálculos es el utilizarla sólo cuando sea imprescindible. En este sentido, George (ver [GeoHer92]) propone calcular con *números en coma flotante* los determinantes $\Delta(x_{i+1}, K_j)$ y con enteros los determinantes $\det(K_j)$. En estos últimos aparecen cantidades tales como $\sum_{i=1}^{d} L_i^d$, cuya evaluación con enteros se puede llevar a cabo si L_i es a lo sumo $2^{16} - 1 = 65.535$. En este caso la resolución es aceptable, pero sigue siendo necesario un gran esfuerzo en tiempo de CPU.

Si se opta por la utilización de aritmética no exacta habrá situaciones en las que sea imposible construir, estrictamente hablando, la triangulación de Delaunay. Lo que sí es factible, es construir una triangulación *próxima* a ésta. Esta posibilidad puede ser más

conveniente para la aplicación del MEF ya que en ocasiones la calidad de la triangulación de Delaunay es peor que la de una triangulación no estrictamente de Delaunay, tal como apuntábamos cuando hablábamos de la aparición de tetraedros planos.

Los problemas causados por los errores de redondeo, así como sus posibles soluciones haciendo uso de aritmética no exacta, han sido tratados por diversos autores, ver por ejemplo [CavFie85], [Perr88], [SchShe90], [WeaHas94]. Los métodos utilizados por estos autores están basados en las siguientes técnicas, o en combinaciones de ellas: disminución o aumento del radio de las esferas circunscritas en una cantidad similar a la máxima precisión con la que opere el ordenador, perturbación virtual de las coordenadas de los puntos que causen problemas, permutación del orden de inserción de estos puntos, eliminación de los tetraedros de D_1^i que posean alguna cara que no sea visible desde el punto x_{i+1}, etc.

Por nuestra parte, hemos probado procedimientos que combinan las tres primeras técnicas y los resultados obtenidos no han sido suficientemente satisfactorios. Esto nos ha llevado a construir un algoritmo, que será explicado más adelante, y que pretende evitar los errores de redondeo en la casi totalidad de los casos.

2.4. Estructuras de datos.

Los objetos matemáticos con que trabaja cualquier algoritmo geométrico no pueden ser manipulados directamente por el ordenador. Por tanto, es necesario simbolizarlos por medio de datos simples susceptibles de ser tratados por éste. La organización de esos datos se conoce como estructura de datos. No existe un único tipo de estructura de datos para representar una malla. Dependiendo de si la malla es estructurada o no estructurada, de la utilización que se le pretenda dar, de la rapidez en el acceso a los datos, la cantidad de memoria disponible, etc., será conveniente un determinado tipo de estructura de datos u otro. Aspectos tan dispares como la representación gráfica, aplicación del MEF mediante solución directa, iterativa o técnicas multimalla, harán que sea apropiada una determinada estructura de datos.

Cualquier estructura de datos debe aportar información suficiente para conocer las coordenadas de los puntos de X y las de los puntos que conforman cada elemento. Además, en el caso de mallas no estructuradas es muy conveniente disponer de algún tipo de información acerca de las conexiones entre elementos. De esta manera podemos conocer los elementos *vecinos* a uno dado, y por tanto, tener una visión global de la malla.

Para la representación de triangulaciones se utiliza con frecuencia una estructura de datos en la que cada punto y cada símplice están especificados en la memoria del ordenador por un nodo, *np* y *ns*, respectivamente. La estructura de datos consta de un vector, $p(i,np)$ $(1 \le i \le d)$ que nos proporciona las d coordenadas cartesianas de cada punto *np*, de otro, $s(i,ns)$ $(1 \le i \le d+1)$, que nos proporciona los nodos de los $d+1$ puntos que forman cada símplice *ns* , y por último, de un puntero $pt(i,ns)$ $(1 \le i \le d+1)$ que *apunta* a cada uno de los símplices *vecinos* del *ns*.

Mediante la utilización adecuada de estos vectores y punteros se pueden conocer los vértices de cualquiera de las l-caras $(1 \le l \le d-1)$ de un determinado símplice, así como el conjunto de l-caras con una $(l-1)$-cara en común [Fort92].

La estructura de datos anterior tiene como ventaja principal el reducido gasto de memoria. Por contra, tiene el inconveniente de no disponer de una numeración de las l-caras de los símplices de la triangulación. Esta información puede ser muy conveniente para la aplicación de algunos programas basados en el MEF.

En el caso particular de la triangulación de Delaunay, es necesario saber cuándo un punto está dentro de la esfera circunscrita a un símplice. En la definición 15 de los preliminares se mostraba un procedimiento, basado en el cálculo un determinante de $(d+1)\times(d+1)$, para establecer cuándo ocurría esta circunstancia. Este procedimiento ofrece la posibilidad de utilizar aritmética exacta; sin embargo, el número de operaciones que conlleva el cálculo de este determinante es muy superior al que se necesitan para determinar el centro y radio de la esfera circunscrita a un símplice, si bien en esta ocasión no es posible utilizar aritmética exacta. Cuando se trabaja con números en coma flotante es preferible esta última alternativa. Además, supone un ahorro de tiempo de CPU el tener almacenados en memoria los datos relativos a estas esferas.

Centrándonos en el caso de dimensión tres, supongamos que v_i $(1 \le i \le 4)$ sean los cuatro vértices de un tetraedro. El centro de la esfera circunscrita a este tetraedro se puede hallar, por ejemplo, calculando la intersección de los tres planos que pasan por los vértices v_1 y v_i y por el punto medio del lado opuesto, el $v_j v_k$, donde se supone que i, j, k toman todos los valores comprendidos entre 2 y 4 con $i \ne j \ne k$. El cálculo de esta intersección supone la resolución de un sistema lineal de ecuaciones de 3×3. El número de operaciones necesario para resolver este sistema, por ejemplo mediante el método de Gausscon pivote total, es muy inferior a las que se necesitarían para resolver el correspondiente determinante de 4×4.

Una estructura de datos para la representación de mallas no estructuradas en dimensión tres que requiere más memoria, pero que a cambio ofrece una información más directa y completa, es la reseñada por George en [Geor91] y denominada *GEOM*. En ella se describen los elementos por sus caras, las caras por sus aristas y las aristas por sus vértices. Aparte de esta información, se puede disponer de datos adicionales como números de referencia asociados a los vértices, etc.

La estructura de datos por la que hemos optado pretende ser lo más versátil y completa posible aún a costa de necesitar gran cantidad de memoria. Otro objetivo que se ha tenido en cuenta es el de la rapidez de funcionamiento de los diferentes algoritmos que componen el programa. Para ello el acceso a la información debe ser lo más directo posible.

Los datos de la triangulación se almacenan en dos tipos de estructuras: *vectores globales* y *vectores locales*. Los vectores globales almacenan información de toda la triangulación construida hasta el momento presente, es decir, de $D(X_i)$. Los vectores locales almacenan datos relativos a la región que se modifica, esto es, D_1^i. Estos datos serán utilizados después en la actualización de los vectores globales.

1. Vectores globales.

En cada fase de la triangulación es necesario conocer el número total de tetraedros, *ntt*, caras, *ntc*, aristas, *nta* y vértices, *ntv*, existentes hasta ese momento. Lógicamente, estos valores cambiando a medida que se vayan añadiendo nuevos puntos a la triangulación.

Hay que tener en cuenta que los vértices de una triangulación conforme coinciden con los puntos de X, por tanto, *ntv* será también el número total de puntos incluidos en la triangulación.

La descripción de los tetraedros se hace a partir de sus caras y sus vértices, la de las caras a partir de sus aristas y sus vértices y la de las aristas a partir de sus vértices. Es claro que los vértices de los tetraedros y caras se pueden deducir a través de los vértices de las aristas. El hecho de optar por esta representación ha sido motivado por tener un acceso más directo, y por tanto más rápido, a los vértices de estos elementos. Se han almacenado también el radio (en realidad, el cuadrado del radio) y las coordenadas del centro de las esferas circunscritas a los tetraedros. Para el cálculo del centro se ha empleado el método de Gauss con pivote total. En el caso de un sistema de 3×3 este método es rápido, y además reduce notablemente los errores de redondeo. En todo lo sucesivo supondremos que *nv*, *na*, *nc* y *nt* son los números representativos de los vértices, aristas, caras y tetraedros.

Coordenadas de los puntos de X:

$$x(nv), y(nv), z(nv).$$

Radio y coordenadas del centro de las esferas circunscritas:

$$r_e(nt), x_e(nt), y_e(nt), z_e(nt).$$

Descripción de los elementos:

1. Descripción del tetraedro por sus caras:

   $$CT(i,nt) = nc \quad (1 \le i \le 4).$$

   El vector CT proporciona las cuatro caras del tetraedro *nt*.

2. Descripción del tetraedro por sus vértices:

   $$VT(i,nt) = nv \quad (1 \le i \le 4).$$

   El vector VT proporciona los cuatro vértices del tetraedro *nt*.

3. Descripción de la cara por sus aristas:

$AC(i,nc) = na \quad (1 \le i \le 3)$.

El vector AC proporciona las tres aristas de la cara nc.

4. Descripción de la cara por sus vértices:

$VC(i,nc) = nv \quad (1 \le i \le 3)$.

El vector VC proporciona los tres vértices de la cara nc.

5. Descripción de la arista por sus vértices:

$VA(i,na) = nv \quad (1 \le i \le 2)$.

El vector VA proporciona los dos vértices de la arista nc. □

Los vectores x, y, z, r_e,, x_e, y_e, y z_e se representan en el ordenador mediante reales de doble precisión, esto es, *números en coma flotante* de dieciséis dígitos. El resto de vectores están representados por enteros de ocho dígitos.

Con la información anterior tenemos un conocimiento completo pero local de los diversos elementos de la triangulación. La estructura de datos se debe completar indicando las conexiones existentes entre ellos. Su utilidad es doble: por un lado proporcionan información de gran interés en la aplicación a un programa de elementos finitos y por otro agilizan el proceso de construcción de la triangulación.

Conexiones entre elementos:

1. Tetraedros que comparten una cara:

$TCC(i,nc) = nt \quad (1 \le i \le ttcc(nc))$.

El vector *TCC* nos proporciona los tetraedros, *nt*, que comparten la cara *nc*. El vector *ttcc* indica el total de tetraedros adyacentes a la cara *nc*. Este vector sólo puede tomar los valores 1 ó 2; el valor 1 lo tomará cuando la cara sea exterior a la triangulación. En realidad, se podría prescindir de este último vector, si acordáramos, por ejemplo, asignar el valor 0 a *TCC* cuando no existiera el correspondiente tetraedro.

2. Caras que comparten una arista:

$$CCA(i,na) = nc \quad (1 \le i \le tcca(na)).$$

El vector *CCA* nos proporciona las caras, *nc*, que comparten la arista *na*. El vector *tcca* indica el total de caras adyacentes a la arista *na*.

3. Aristas que comparten un vértice:

$$ACV(i,nv) = na \quad (1 \le i \le tacv(nv)).$$

El vector *ACV* nos proporciona las aristas, *na*, que comparten el vértice *nv*. El vector *tacv* indica el total de aristas adyacentes al vértice *nv*. □

2. Principales vectores locales.

Cuando se incluye el punto x_{i+1} en la triangulación, los tetraedros de D_1^i desaparecen. Todas las caras y aristas de estos tetraedros, cuyo interior relativo esté incluido en el interior de D_1^i, desaparecen también. Denotaremos por $C_{iD_1}^i$ y $A_{iD_1}^i$ a estas caras y aristas, respectivamente.

La reconstrucción de la triangulación se lleva a cabo a partir de las caras, aristas y vértices incluidos en la frontera de D_1^i y el punto x_{i+1}. Denotaremos por $C_{fD_1}^i$, $A_{fD_1}^i$ y $V_{fD_1}^i$ a las caras, aristas y vértices de la frontera de D_1^i.

El conjunto de tetraedros que se forman con las caras de $C^i_{fD_1}$ y el punto x_{i+1} se denota por D^i_2.

$$D^i_2 = \bigcup_j \text{conv}(C_j \cup \{x_{i+1}\}) \quad \forall C_j \in C^i_{fD_1}$$

El conjunto de caras que se forman con las aristas de $A^i_{fD_1}$ y el punto x_{i+1} se denota por C^i_2.

$$C^i_2 = \bigcup_j \text{conv}(A_j \cup \{x_{i+1}\}) \quad \forall A_j \in A^i_{fD_1}$$

Por último, el conjunto de aristas formadas con los vértices de $V^i_{fD_1}$ y el punto x_{i+1} se denota por A^i_2.

$$A^i_2 = \bigcup_j \text{conv}(V_j \cup \{x_{i+1}\}) \quad \forall V_j \in V^i_{fD_1}$$

La estructura de datos almacena en los vectores locales los números *globales* de los elementos interiores y de la frontera de D^i_1, así como los construidos a partir de ésta. Entendemos por número *global* aquel que representa al elemento en la triangulación $D(X_i)$. El índice de un determinado elemento dentro de su vector local es lo que entenderemos por *número local* del elemento.

En situaciones especiales pude ocurrir que al incluir un punto en la triangulación se formen menos tetraedros, caras o aristas de las que han desaparecido. Ésto da lugar a la aparición de lo que denominamos, elementos residuales. Cuando esto ocurre, se hace necesario almacenar los números globales que no hayan sido reasignados a nuevos elementos. Los vectores empleados para este propósito son los mismos que se emplean para almacenar D^i_1, $C^i_{iD_1}$ y $A^i_{iD_1}$, que como veremos son *LT*, *LCI* y *LAI*, respectivamente. Para ilustrar el proceso de almacenamiento de elementos residuales tomaremos como ejemplo el vector *LT*. Consideremos que el número total de tetraedros que hay acumulados en el vector *LT*, cuando se ha incluido el punto x_i en la triangulación, es *nrt*.

Supongamos que en el momento de incluir el punto x_{i+1} el conjunto D_1^i consta de n tetraedros. En el vector LT se almacenarán estos tetraedros a partir de $nrt+1$ hasta $tt = n - nrt$ (figura 12 (a)). Si, por ejemplo, ocurre que el cardinal del conjunto $C^i_{fD_1}$, tcf, es menor que tt (ver figura 12 (b)), los tetraedros cuyo número local esté comprendido entre $tcf+1$ y $tt-tcf$, quedarán a la espera de aportar su número global a los tetraedros que se irán generando a medida que se añadan nuevos puntos a la triangulación; su almacenamiento se llevará a cabo situándolos en la cola del vector LT (figura 12 (c)). El puntero nrt tomará el valor $tt-tcf$ hasta su nueva actualización. Si, por contra, tcf es mayor que tt, todos los tetraedros almacenados en LT aportarán su número global a los tetraedros generados en esta fase de la triangulación. En este caso $nrt = 0$. Con los vectores LCI y LAI se procede de forma similar; en este caso se utilizan los punteros nrc y nra para las caras y aristas, respectivamente.

Hay que hacer notar que los números totales nta, ntc y ntt, a los que nos hemos referido con anterioridad, incluyen también las aristas, caras y tetraedros residuales presentes en una determinada fase del programa.

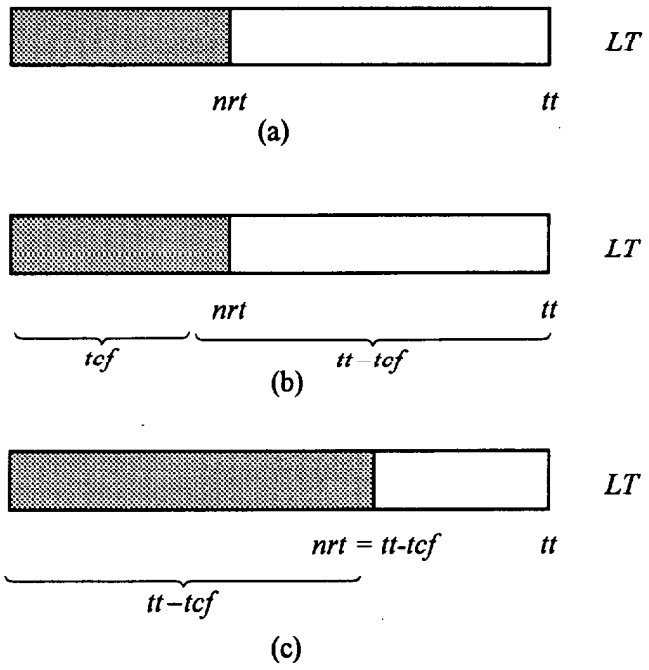


Figura 12. Almacenamiento de tetraedros en LT.

1. Tetraedros residuales y tetraedros de D_1^i.

   $LT(i) = nt \quad (1 \le i \le tt)$.

   Desde $i = 1$ hasta *nrt*, el vector *LT* nos proporciona el número global del *i*-ésimo tetraedro residual. Desde $i = nrt + 1$ hasta *tt*, *LT* nos proporciona el *i*-ésimo tetraedro de D_1^i.

2. Caras residuales y caras de $C_{iD_1}^i$.

   $LCI(i) = nc \quad (1 \le i \le tci)$.

   Desde $i = 1$ hasta *nrc*, el vector *LCI* nos proporciona el número global de la *i*-ésima cara residual. Desde $i = nrc + 1$ hasta *tci*, *LCI* nos proporciona la *i*-ésima cara de $C_{iD_1}^i$.

3. Aristas residuales y aristas de $A_{iD_1}^i$.

   $LAI(i) = na \quad (1 \le i \le tai)$.

   Desde $i = 1$ hasta *nra*, el vector *LAI* nos proporciona el número global de la *i*-ésima arista residual. Desde $i = nra + 1$ hasta *tai*, *LAI* nos proporciona la *i*-ésima arista de $A_{iD_1}^i$.

4. Caras de $C_{fD_1}^i$.

   $LCF(i) = nc \quad (1 \le i \le tcf)$.

   El vector *LCF* nos proporciona el número global de la cara *i*-ésima $C_{fD_1}^i$.

5. Aristas de $A_{fD_1}^i$.

   $LAF(i) = na \quad (1 \le i \le taf)$.

El vector LAF nos proporciona el número global de la arista i-ésima de $A^i_{fD_1}$.

6. Vértices de $V^i_{fD_1}$.

$$LVF(i) = nv \quad (1 \le i \le tvf).$$

El vector LVF nos proporciona el número global del vértice i-ésimo de $V^i_{fD_1}$.

7. Tetraedros de D^i_2.

$$LT2(i) = nt \quad (1 \le i \le tcf).$$

El vector $LT2$ nos proporciona el i-ésimo tetraedro de D^i_2.

8. Caras de C^i_2.

$$LC2(i) = nc \quad (1 \le i \le taf).$$

El vector $LC2$ nos proporciona la i-ésima cara de C^i_2.

9. Aristas de A^i_2.

$$LA2(i) = na \quad (1 \le i \le tvf).$$

El vector $LA2$ nos proporciona la i-ésima arista de A^i_2. □

2.5. Desarrollo del programa de triangulación.

En líneas generales, el algoritmo de triangulación que hemos empleado corresponde al descrito en el apartado 2.2. Se ha adoptado la simplificación, comentada al final de dicho apartado, consistente en incluir todos los puntos de X dentro de un cubo previamente descompuesto en tetraedros.

A continuación (figura 13) se muestra el diagrama de flujo del programa de triangulación, denominado MALLA. Cada vez que se añade un nuevo punto a la triangulación se tiene que actualizar la estructura de datos a la que nos hemos referido en el apartado anterior. Los aspectos más relevantes de cada una de las subrutinas que componen el programa se explicarán en los apartados siguientes.

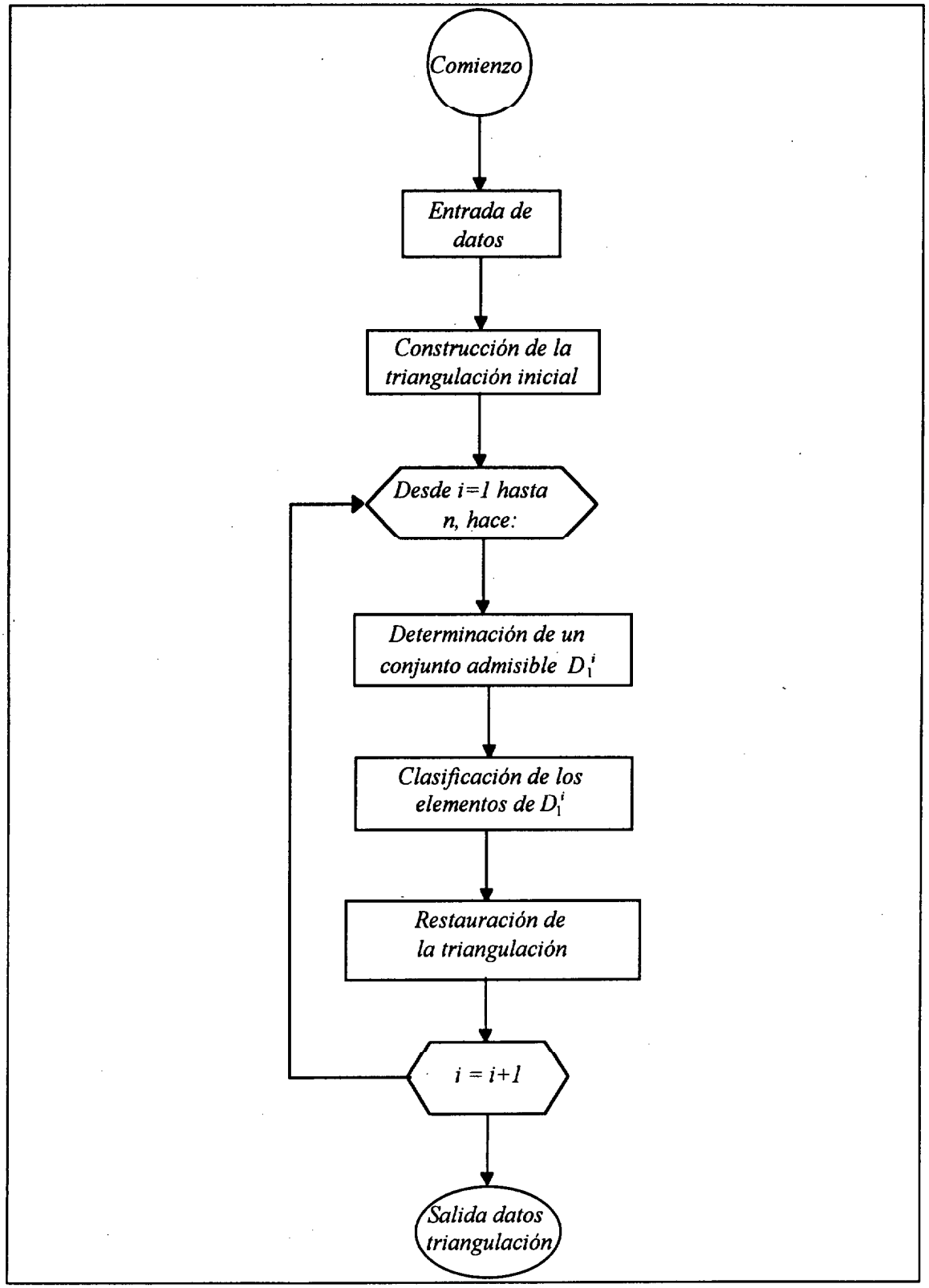


Figura 13. Diagrama de flujo del programa de triangulación.

2.5.1. Entrada de datos.

La entrada de datos consiste en una lista de puntos, $X \subset E^3$, definidos por sus coordenadas. Las coordenadas de estos puntos se almacenan en los vectores $x(nv)$, $y(nv)$ y $z(nv)$. El programa comienza leyendo del fichero de entrada el número asociado a cada punto y sus correspondientes coordenadas. La definición de los puntos se hace atendiendo a dos criterios fundamentales: calidad de la triangulación y conformidad entre la triangulación y la frontera del dominio. De estos aspectos nos ocuparemos en el capítulo tercero. Por el momento nos centraremos en el problema de construir la triangulación de Delaunay un conjunto de puntos previamente definido.

2.5.2. Construcción de la triangulación inicial.

Como se comentó anteriormente, para simplificar el algoritmo de triangulación, se construye un cubo previamente descompuesto en cinco tetraedros. Todos los vectores de la estructura de datos se *inicializan* con los datos de esta triangulación inicial.

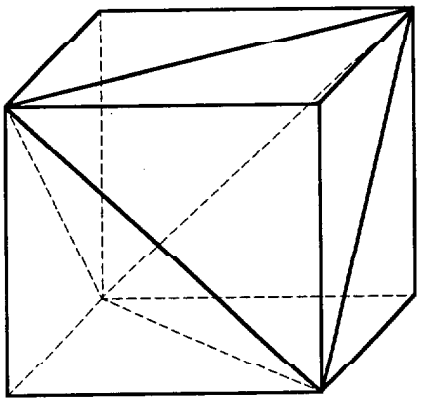


Figura 14. Triangulación inicial del cubo que contiene a X.

Las coordenadas de los vértices del cubo se ajustan para conseguir que todos los n puntos de X se encuentren en el interior del mismo. Para ello, hallamos el *centro de gravedad* de la nube de puntos:

$$\vec{x}_c = \frac{1}{n}\sum_{i=1}^{n} \vec{x}_i$$

donde $\vec{x}_i$ son los vectores de posición de los puntos de X y $\vec{x}_c$ es el vector de posición del *centro de gravedad* de X. El radio r de la nube de puntos se halla calculando el máximo de todas las distancias entre los puntos y el centro.

$$r = \max_{1 \le i \le n} \{ \mathrm{d}(x_i, x_c) \}$$

Las coordenadas de los vértices del cubo se calculan de forma que la esfera de centro x_c y de radio r esté contenida dentro del mismo. Las coordenadas $x^j_{i\,cub}$ $(1 \le i \le 8)$ de los vértices del cubo están dados por:

$$x^j_{i\,cub} = x^j_{i\,c} \pm \delta \cdot r_c \quad \delta > 1$$

donde el signo más o menos se aplica convenientemente para conseguir que estas coordenadas definan un cubo. El parámetro δ se introduce para interponer una distancia aceptable entre la nube de puntos y el cubo. De esta manera se evita que los tetraedros construidos con las caras exteriores del cubo sean muy degenerados.

Si el número de puntos de X es elevado, la cantidad de tetraedros que tienen como vértice uno de los vértices del cubo puede ser muy grande, ésto hace que esos tetraedros tengan uno de sus ángulos sólidos muy pequeño. Los errores de redondeo en los cálculos que involucren a estos tetraedros aumentarán a medida que estos ángulos disminuyan. Además, la cantidad de memoria disponible para la variable *ACV*, que almacena las aristas compartidas por un vértice, deberá aumentar en consecuencia. Por todo esto, es recomendable incluir en X un conjunto de puntos X_e distribuidos sobre una superficie esférica interior al cubo. Los puntos de X_e deben estar situados entre el cubo y la nube de puntos X a una distancia apropiada para que se conserve la conformidad entre la triangulación y la frontera del dominio. Como valores típicos cabe decir que el número

de puntos de X_e que se introducen en el programa es del orden de 60 para un conjunto X de unos 10^4 puntos, la distancia a x_c es $2 \cdot r$ y el parámetro δ toma el valor 3. Para simplificar la notación, consideraremos a partir de ahora que $X := X \cup X_e$.

2.5.3. Determinación de D_1^i.

La parte más crítica del programa es la que se encarga de determinar cuáles son los tetraedros que componen el conjunto D_1^i. Un cálculo inapropiado de este conjunto puede traer como consecuencia, tal como se explicó en el apartado dedicado a errores de redondeo, una construcción que topológicamente no corresponda a una triangulación. El resto de procesos que intervienen en el programa son una mera clasificación y actualización de los vectores almacenados en la estructura de datos. Si el conjunto D_1^i es *admisible*, el programa funcionará sin problemas. Para establecer el significado preciso, dentro de este contexto, de la palabra *admisible* debemos dar unas definiciones previas.

Definición 19.

Diremos que una cara $C \in C_{fD_1}^i$ es ε-visible desde x_{i+1} si ésta es visible y además sucede que:

$$\min_{x \in C} \{\cos(\overline{x_{i+1}x}, \vec{n})\} > \varepsilon$$

para cierto parámetro $\varepsilon > 0$, donde $\vec{n}$ es el vector unitario normal a la cara C y saliente de D_1^i. Es claro que este mínimo ocurre cuando x es uno de los vértices de la cara C. □

Definición 20.

Diremos que un politopo (poliedro acotado) es ε-en forma de estrella si todas sus caras son ε-visibles. □

Los requisitos que debe satisfacer el conjunto D_1^i son los siguientes:

1. Un tetraedro T pertenece a D_1^i sólo si $x_{i+1} \in B(T)$.
2. El conjunto D_1^i debe ser ε-en forma de estrella.
3. El conjunto D_1^i no debe contener ningún punto de X aparte de x_{i+1}. □

El parámetro ε debe ser, por una parte, suficientemente grande como para asegurar que el programa sea capaz de trabajar con aritmética no exacta, por ejemplo con reales de doble precisión, y por otra, suficientemente pequeño como para que la triangulación resultante se aproxime a la de Delaunay. Puede ocurrir que el valor asignado inicialmente a ε sea demasiado restrictivo, es decir, que no se pueda construir un conjunto D_1^i que cumpla las condiciones anteriores, en concreto la segunda. Cuando esto ocurre, el programa cambia automáticamente su valor por otro más pequeño.

La segunda condición juega un importante papel en el evitar que aparezcan tetraedros muy planos. De hecho, la calidad de la triangulación que resulta de imponer los requisitos anteriores puede ser mejor, como se comentó en el apartado 2.3, que la calidad de la correspondiente triangulación de Delaunay.

Los pasos principales que se siguen para determinar D_1^i son:

- Búsqueda del tetraedro o conjunto de tetraedros que contienen a x_{i+1}. A este conjunto de tetraedros, N^i, lo denominaremos *núcleo*.
- Adición de tetraedros al núcleo, con ciertos controles, hasta construir un conjunto D_1^i admisible.

Localización y determinación del núcleo.

Como hemos dicho anteriormente, el núcleo es el tetraedro o conjunto de tetraedros que contienen al punto x_{i+1}. Para hallar uno de estos tetraedros nos desplazamos a través de la triangulación en la dirección de dicho punto. La búsqueda del núcleo puede comenzar en cualquiera de los tetraedros de $D(X_i)$, aunque como es

lógico, ésta será más corta si el tetraedro del que se parte está cerca del punto x_{i+1}. En la estructura de datos no tenemos ninguna información que nos permita relacionar el número asignado a cada tetraedro con su posición espacial, por tanto, no podemos saber a priori si un tetraedro está cerca o lejos x_{i+1}. No obstante, debido a la forma ordenada en que se han creado los puntos de X, cabe esperar que normalmente el punto x_{i+1} esté próximo al último tetraedro incluido en $D(X_i)$. Elegiremos este último tetraedro, T_u, para empezar la búsqueda del núcleo. El algoritmo de búsqueda consiste en pasar de un tetraedro, T, al tetraedro adyacente, T_v, que esté en la dirección de "la recta" que une T con x_{i+1} y en el sentido de aproximación al punto. El éxito del procedimiento está garantizado por el hecho de que la triangulación constituye un conjunto convexo, y por tanto, siempre se puede encontrar el tetraedro adyacente T_v al que nos referíamos anteriormente. A pesar de que el camino seguido para encontrar el núcleo es el más recto posible eso no garantiza que sea el de mínimo coste computacional, es decir, el que menos tetraedros atraviesa hasta llegar al núcleo; vease por ejemplo la figura 15. Si la estructura de la triangulación es suficientemente regular en cuanto al tamaño y forma de los tetraedros, es bastante probable que el camino recto sea uno de los de menor coste computacional. Sin embargo, con el conocimiento que se tiene de la triangulación, no es posible saber a priori cual es camino óptimo.

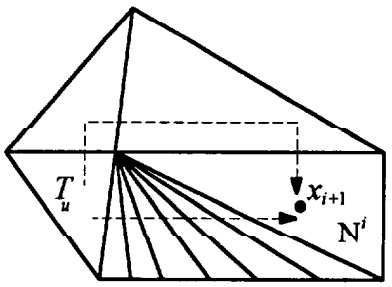


Figura 15. Camino óptimo y camino recto para encontrar el núcleo.

El procedimiento BÚSQUEDA localiza, inicialmente, uno de los tetraedros del núcleo (ver figura 16). Una vez hecho esto, utiliza el procedimiento NÚCLEO para hallar el resto.

El procedimiento BÚSQUEDA localiza, inicialmente, uno de los tetraedros del núcleo (ver figura 16). Una vez hecho esto, utiliza el procedimiento NÚCLEO para hallar el resto.

Algoritmo para la localización del núcleo.

procedimiento BÚSQUEDA

entrada: T_u.

(* *comienzo inicialización de variables* *)

$T := T_u$

fin$:=$ *falso*

(* *final inicialización de variables* *)

mientras (*fin* = *falso*) **hace**

cosmax$:= -1$

desde $j := 1$ **hasta** 4 **hace** (* *bucle en caras* *)

hallamos la cara $C_j \in T$

hallamos un vértice, x_{cj} *, de la cara* C_j

hallamos el vector unitario, $\vec{v}$*, en la dirección de* x_{cj} *a* x_{i+1}

hallamos el vector, $\vec{n}$*, normal a la cara* C_j *unitario y saliente de* T

si $(\vec{n} \cdot \vec{v} > cosmax)$ **entonces** (* *se busca la cara C que "mejor vea" a* x_{i+1} *)

$cosmax := \vec{n} \cdot \vec{v}$

$C := C_j$

fin si

$d_j := \vec{n} \cdot \overline{x_{cj} x_{i+1}}$

fin desde

$d := \max_{1 \le j \le 4} \{d_j\}$

si $(d \le \delta)$ **entonces** (* *criterio de parada* *)

llamamos al **procedimiento** NÚCLEO (* *el tetraedro T pertenece al núcleo* *)

fin:= *verdadero*

si no (* *pasamos al tetraedro vecino* *)

hallamos el tetraedro $T_v \neq T / C \in T_v$

$T := T_v$

fin si

fin mientras

salida: N^i, C^i_{iN}.

fin procedimiento.

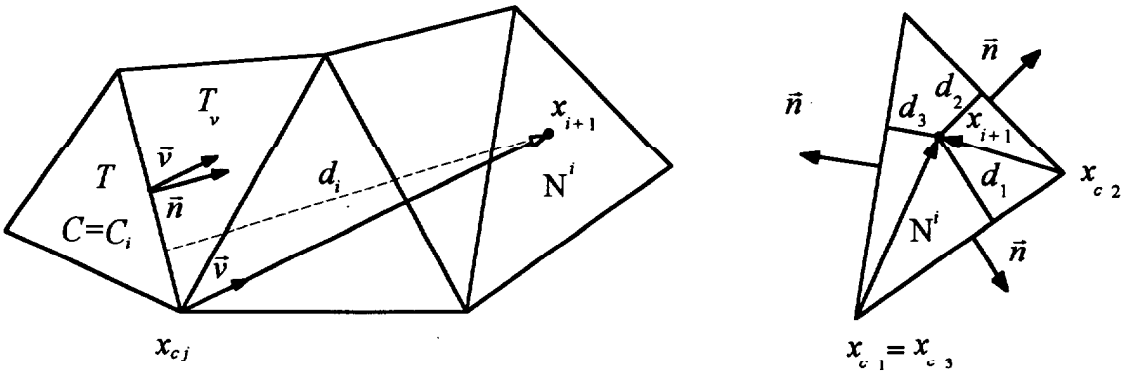


Figura 16. Búsqueda del núcleo (a). *Criterio de parada del algoritmo* (b).

En el criterio de parada del algoritmo se ha puesto la condición $d \leq \delta$, donde se supone que δ es un número positivo pequeño similar a la precisión del ordenador, en vez de $d \leq 0$ como parecería lógico. La razón para hacer esto se encuentra en que el valor obtenido para d, cuando el punto está sobre una cara común a dos tetraedros, digamos T y T', depende ligeramente de que los cálculos se realicen estando situados en T o en T'. Podría ocurrir que, estando situados en T o en T', el valor calculado para d fuera positivo (del orden de δ). Esto traería como consecuencia el paso alternativo de un

tetraedro a otro sin parar en ninguno. Con la condición $d \leq \delta$ se evita esta oscilación. El valor asignado a δ, cuando se trabaja en doble precisión, es del orden de 10^{-12}.

Una vez localizado uno de los tetraedros del núcleo, le corresponde al procedimiento NÚCLEO hallar los demás, si los hay. Además, NÚCLEO se encarga de hallar el parámetro ε_N, que es el máximo valor para el que N^i es ε-en forma de estrella respecto del punto x_{i+1}.

Como se verá más adelante, el procedimiento CONJUNTO D_1^i necesita conocer cuáles son las caras interiores a N^i. A este conjunto lo denominaremos C_{iN}^i y será hallado también por el procedimiento NÚCLEO.

Algoritmo de construcción del núcleo.

procedimiento NÚCLEO

entrada: $T, \{C_j\}, \{d_j\}, x_{i+1}$.

(* *comienzo inicialización de variables* *)

$k := 0$

$\varepsilon_N := 1$

(* *final inicialización de variables* *)

desde $j := 1$ **hasta** 4

si $(d_j \leq \delta')$ **entonces** (* *criterio para determinar cuando* x_{i+1} *está "sobre" una cara* *)

$k := k + 1$

$C_k' := C_j$ (* *almacenamos en memoria las caras próximas a* x_{i+1} *)

fin si

fin desde

si $(k = 0)$ **entonces** (* *si no hay ninguna cara próxima a* x_{i+1} *el núcleo contiene un único tetraedro* *)

$\mathrm{N}^i := \{T\}$

$\mathrm{C}^i_{i\mathrm{N}} := \varnothing$

fin si

si $(k = 1)$ **entonces** (* *si sólo hay una cara próxima a* x_{i+1} *el núcleo contiene a los dos tetraedros que la comparten* *)

$\mathrm{N}^i := \{T_v / C'_1 \in T_v\}$

$\mathrm{C}^i_{i\mathrm{N}} := \{C'_1\}$

fin si

si $(k = 2)$ **entonces** (* *si hay dos caras próximas a* x_{i+1} *el núcleo contiene a todos los tetraedros que comparten la arista común a ambas caras* *)

hallamos la arista $A / A = C'_1 \bigcap C'_2$

$\mathrm{N}^i := \{T_v / A \in T_v\}$

$\mathrm{C}^i_{i\mathrm{N}} := \{C / A \in C\}$

fin si

si $(k > 2)$ **entonces** (* *el tetraedro* T *es extremadamente degenerado, el tratamiento que efectuamos es similar al caso* $k = 2$ *)

hallamos los dos valores, k_1 *y* k_2*, que minoran* $\{d_j\}$

hallamos la arista $A / A = C'_{k_1} \bigcap C'_{k_2}$

$\mathrm{N}^i := \{T_v / A \in T_v\}$

$\mathrm{C}^i_{i\mathrm{N}} := \{C / A \in C\}$

fin si

hallamos el conjunto $\mathrm{C}^i_{f\mathrm{N}}$ *de caras de la frontera de* N^i

desde $j := 1$ **hasta** $\mathrm{card}(\mathrm{C}^i_{f\mathrm{N}})$

hallamos la cara $C_j \in \mathrm{C}^i_{f\mathrm{N}}$;

hallamos el vector, $\vec{n}$*, normal a la cara* C_j*, unitario y saliente de* N^i

cosmin$:= \min_{1 \le l \le 3} \{\cos(\overline{x_{i+1} x_{cl}}, \vec{n})\}$ *donde* x_{cl} *es el vértice l-ésimo de la cara* C_j

si $(cosmin < \varepsilon_{\mathrm{N}})$ **entonces**

Los casos con $k = 3$ y $k = 4$ corresponden a tetraedros para los que el punto x_{i+1} está muy cerca (a una distancia inferior a δ') de tres o cuatro de sus caras, es decir, tetraedros muy degenerados. Como consecuencia, el núcleo estará muy mal conformado, aunque en la práctica esta circunstancia raramente se produce.

El procedimiento NÚCLEO considera que x_{i+1} está sobre una cara o una arista cuando en realidad está a una distancia pequeña, δ', muy inferior a la distancia entre puntos de X. Si, por ejemplo, el criterio para establecer cuando x_{i+1} está sobre una cara, fuera $d_j \leq 0$, podría ocurrir que x_{i+1} estuviera en el interior de un tetraedro T, pero muy próximo a una de sus caras, C. En ese caso el procedimiento NÚCLEO consideraría que $N^i = \{T\}$. El parámetro ε_N asociado a este núcleo sería excesivamente pequeño. Está claro que este resultado se mejoraría considerando al núcleo formado por los dos tetraedros que comparten C. Esta circunstancia se consigue adoptando el criterio $d_j \leq \delta'$ en vez de $d_j \leq 0$. El valor que asignamos a δ', trabajando con doble precisión, es del orden de 10^{-8}.

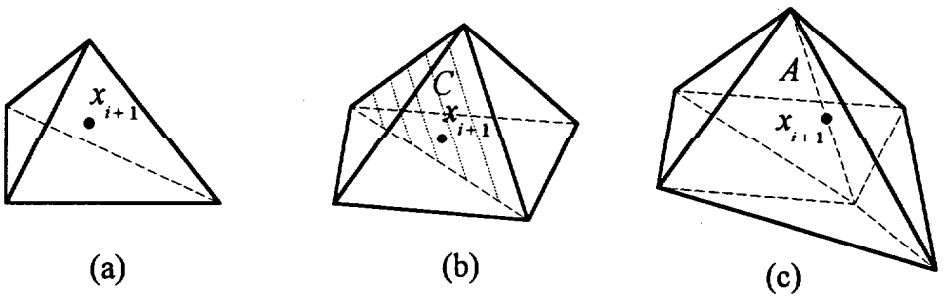


Figura 17. Núcleo formado por un sólo tetraedro (a), *por los tetraedros que comparten la cara C* (b) *y por los que comparten la arista A* (c).

Construcción de D_1^i.

La construcción de D_1^i se hace por adición, con ciertos controles, de tetraedros al núcleo. El algoritmo CONJUNTO D_1^i realiza una serie de iteraciones en cada una de las cuales añade algunos tetraedros a D_1^i, conjunto que previamente se ha hecho coincidir con N^i. Los tetraedros que se añaden en cada iteración son almacenados en el conjunto

$ND_1'^i$. Al final de cada una de ellas, se igualan los conjuntos $ND_1'^i$ y ND_1^i, y D_1^i se actualiza. En la siguiente iteración se realiza un recorrido por las caras de cada tetraedro T de ND_1^i, y se analiza si la esfera circunscrita al tetraedro T_v, exterior a D_1^i y tal que $C = T \cap T_v$, contiene al punto x_{i+1}. Si este es el caso, el tetraedro T_v se añade a D_1^i; en caso contrario, se analiza si C es ε-visible desde x_{i+1}. Si es así, el algoritmo prosigue; si no, se marca el tetraedro T (marcando todas sus caras) de manera que éste sea considerado por el algoritmo como si $x_{i+1} \notin B(T)$, y CONJUNTO D_1^i comienza desde el principio. El procedimiento acaba cuando en una de las iteraciones no se añade ningún nuevo tetraedro, esto es, cuando $ND_1'^i = \varnothing$.

El recorrido a través de las caras de ND_1^i se utiliza para clasificar las caras de D_1^i en caras interiores, $C_{iD_1}^i$, y caras de la frontera, $C_{fD_1}^i$. Estos conjuntos se utilizarán en la renovación de la triangulación.

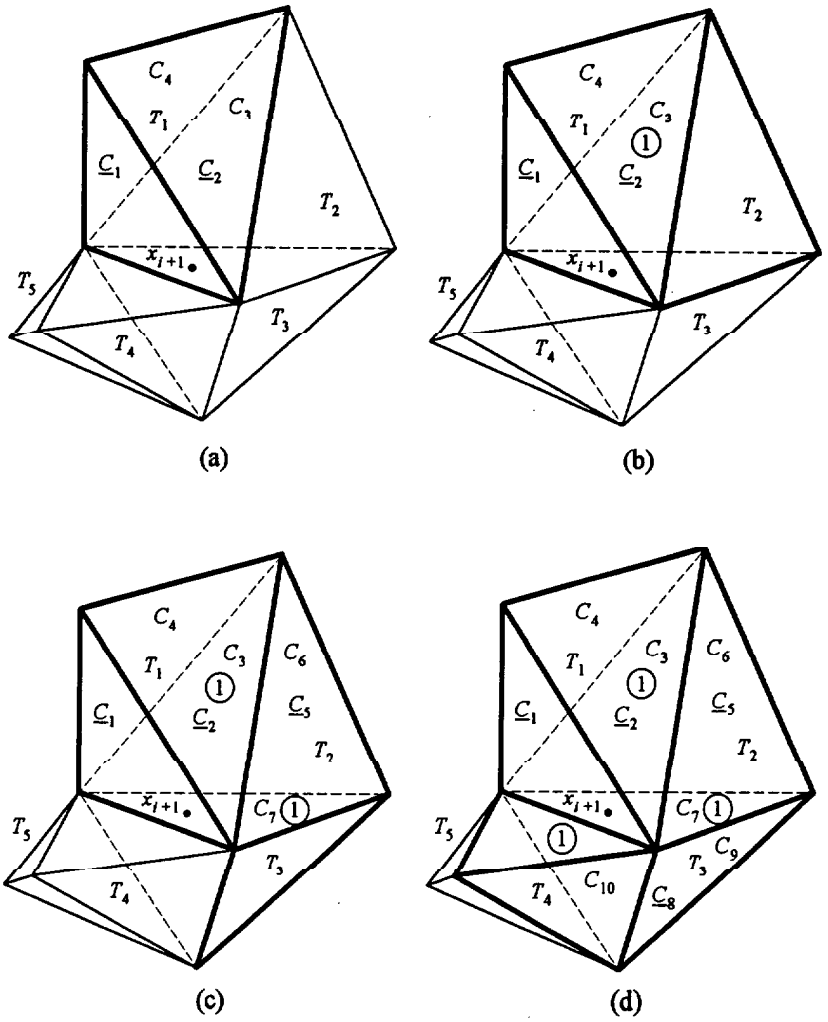
Habrá caras que hayan sido consideradas en alguna de las iteraciones previas. Para evitar repeticiones en la inclusión de tetraedros a D_1^i o caras a $C_{iD_1}^i$, se marcan las caras una vez que se hayan incluido en $C_{iD_1}^i$. Para marcarlas se utiliza un vector $M(C)$. Este vector es igual a 0 si la cara C no ha sido incluida en $C_{iD_1}^i$, en caso contrario, valdrá 1. Este vector será utilizado también para marcar las caras de los tetraedros que tienen alguna de sus caras no ε-visible desde x_{i+1}. En este caso, la marca utilizada será -1.

Puede darse el caso de que un tetraedro T tenga una cara de $C_{fD_1}^i$ no ε-visible y que pertenezca al núcleo. En esta situación, el actual valor de ε es demasiado restrictivo (un valor inicial típico para ε es 0,002). Como es sabido, el núcleo es ε-en forma de estrella para $\varepsilon = \varepsilon_N$, será suficiente cambiar ε por ε_N para asegurar que el proceso concluya.

Una elección más conservadora es cambiar ε por una secuencia decreciente de valores de ε_i, tales que $\varepsilon_N \leq \varepsilon_i < \varepsilon$. Esta elección, probablemente, consigue construir

conjuntos D_1^i más aceptables, pero tiene el inconveniente de tardar más tiempo en converger.

En la figura 18 se ha representado la secuencia utilizada por el algoritmo CONJUNTO D_1^i para construir D_1^i. El conjunto $\{T_1,\ldots,T_5\}$ está formado por todos los tetraedros cuyas esferas circunscritas contienen a x_{i+1}. El núcleo es $N^i = \{T_1\}$ y $C_{iN}^i = \varnothing$ (las caras subrayadas son las que están en primer plano). Las caras con un 1 dentro de un círculo corresponden a las caras para las que $M(C) = 1$.



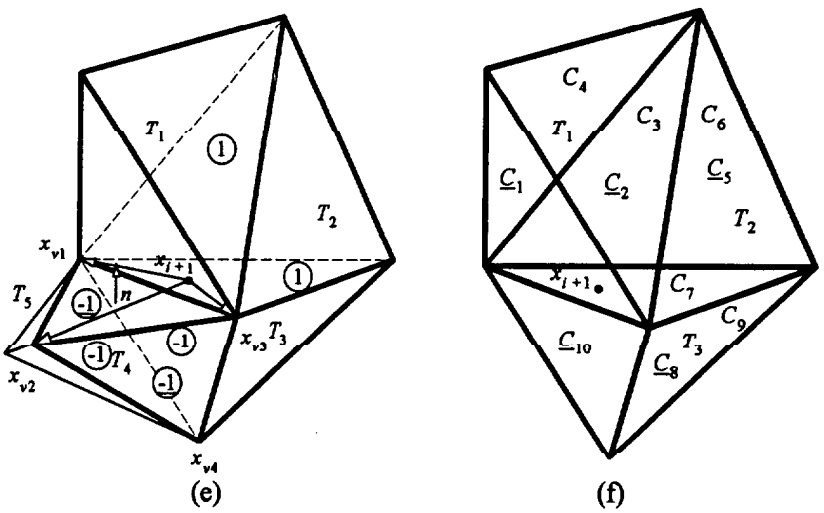


Figura 18. Diferentes pasos del algoritmo CONJUNTO D_1^i.

En la figura 18 se muestran los conjuntos D_1^i, $C_{iD_1}^i$ y $C_{fD_1}^i$ en diferentes pasos del algoritmo CONJUNTO D_1^i. Las caras que están en primer plano están subrayadas.

1. Comienzo del algoritmo: $D_1^i = \{T_1\}$, $C_{iD_1}^i = \varnothing$ y $C_{fD_1}^i = \varnothing$ (a).

2. Primer paso: $D_1^i = \{T_1, T_2\}$, $C_{iD_1}^i = \{C_3\}$ y $C_{fD_1}^i = \{C_1, C_2, C_4\}$ (b).

3. Segundo paso: $D_1^i = \{T_1, T_2, T_3\}$, $C_{iD_1}^i = \{C_3, C_7\}$ y $C_{fD_1}^i = \{C_1, C_2, C_4, C_5, C_6\}$ (c).

4. Tercer paso: $D_1^i = \{T_1, T_2, T_3, T_4\}$, $C_{iD_1}^i = \{C_3, C_7, C_{10}\}$ y $C_{fD_1}^i = \{C_1, C_2, C_4, C_5, C_6, C_8, C_9\}$ (d).

5. Cuarto paso: la cara C no es ε-visible desde x_{i+1}, por ejemplo, para el vértice x_{v2} se tiene que $\cos(\overline{x_{i+1}x_{v2}}, \vec{n}) < 0$. Se marcan todas las caras del tetraedro T_4 con -1 (e).

6. El algoritmo comienza de nuevo la determinación de D_1^i. Ahora, la esfera circunscrita al tetraedro T_4 se considera como si tuviera radio nulo, y por tanto, no contuviera x_{i+1}. Esta es la razón por la que el algoritmo se detiene

una vez que T_4 ha sido añadido a D_1^i. Los conjuntos resultantes son: $\mathrm{D}_1^i = \{T_1, T_2, T_3\}$, $\mathrm{C}_{f\mathrm{D}_1}^i = \{C_1, C_2, C_4, C_5, C_6, C_8, C_9, C_{10}\}$ y $\mathrm{C}_{i\mathrm{D}_1}^i = \{C_3, C_7\}$ (f). □

Como puede verse en la figura 18, no es posible determinar un conjunto D_1^i admisible sin más que extraer T_4 de $\{T_1, \cdots, T_5\}$. El conjunto resultante no sería conexo.

Como se comentó anteriormente, el algoritmo CONJUNTO D_1^i utiliza un sistema de marcas en las caras de los tetraedros. Según la fase del proceso en la que nos encontremos será necesario que las marcas de las caras estén *inicializadas* de una determinada manera; por este motivo, utilizaremos dos procedimientos de inicialización diferentes: INICIALIZACIÓN 1 e INICIALIZACIÓN 2. Por considerar que no tiene mayor interés el detallar como se realizan estos procesos, nos limitaremos a explicar como quedan las marcas de las caras a la salida de cada uno de estos procedimientos.

procedimiento INICIALIZACIÓN 1

Si denominamos $M(C)$ a la marca de la cara C, a la salida del procedimiento las caras deben estar marcadas de la forma siguiente:

$$M(C) = \begin{cases} 1 & \text{si } C \in \mathrm{C}_{i\mathrm{N}}^i \\ 0 & \text{si } C \notin \mathrm{C}_{i\mathrm{N}}^i \end{cases}$$

procedimiento INICIALIZACIÓN 2

A la salida de este procedimiento las caras deben estar marcadas de la forma siguiente:

$$M(C) = \begin{cases} 1 & \text{si } C \in \mathrm{C}_{i\mathrm{N}}^i \\ -1 & \text{si } C \notin \mathrm{C}_{i\mathrm{N}}^i \text{ y } M(C) = -1 \text{ previamente} \\ 0 & \text{si } C \notin \mathrm{C}_{i\mathrm{N}}^i \text{ y } M(C) \neq -1 \text{ previamente} \end{cases}$$

Algoritmo de construcción de D_1^i.

Preproceso:

llamada al **procedimiento** INICIALIZACIÓN 1

Proceso:

procedimiento CONJUNTO D_1^i

entrada: $\varepsilon, \varepsilon_N, M(C), N^i, C_{iN}^i$.

(* *comienza la inicialización de conjuntos* *)

$D_1^i := N^i$

$C_{iD_1}^i := C_{iN}^i$

$ND_1^i := N^i$

$C_{fD_1}^i := \varnothing$

$ND_1'^i := \varnothing$

(* *finaliza la inicialización de conjuntos* *)

mientras $(\text{card}(ND_1^i) > 0)$ **hace**

desde $j := 1$ **hasta** $\text{card}(ND_1^i)$ **hace** (* *bucle en tetraedros de* ND_1^i *)

hallamos el tetraedro $T_j \in ND_1^i$

desde $k := 1$ **hasta** 4 **hace** (* *bucle en las caras del tetraedro* T_j *)

hallamos la cara $C_k \in T_j$

si $(M(C_k) = 0 \text{ o } M(C_k) = -1)$ **entonces** (* *sólo consideramos las caras por las que no hemos "pasado" antes* *)

si $(\exists\, T \neq T_j \,/\, C_k \in T)$ **entonces** (* *la cara es interior al prisma inicial* *)

si $(x_{i+1} \in R(T)$ y $M(C_k) = 0)$ **entonces** (* *el tetraedro T pertenece a* D_1^i *)

si $(\exists C \in T / M(C) = 1)$ **entonces** (* *el tetraedro T ya ha sido incluido en* D_1^i. *No se incluye de nuevo* *)

(* *comienza la actualización de conjuntos* *)

$M(C_k) := 1$

$C_{iD_1}^i := C_{iD_1}^i \cup C_k$

(* *finaliza la actualización de conjuntos* *)

si no (* *el tetraedro T no ha sido incluido en* D_1^i, *lo incluimos ahora* *)

(* *comienza la actualización de conjuntos* *)

$ND_1'^i := ND_1'^i \cup T$

$D_1^i := D_1^i \cup T$

$M(C_k) := 1$

$C_{iD_1}^i := C_{iD_1}^i \cup C_k$

(* *finaliza la actualización de conjuntos* *)

fin si

si no (* *tetraedro T no puede pertenecer a* D_1^i *)

hallamos el vector, $\vec{n}$, *normal a la cara* C_k, *unitario y saliente de* T_j

calculamos cosmin: $= \min_{1 \le l \le 3} \{\cos(\overline{x_{i+1}x_{cl}}, \vec{n})\}$, *donde* x_{cl} *es el vértice l-ésimo de la cara* C_k.

si $(cosmin > \varepsilon)$ **entonces** (* *la cara* C_k *es* ε*-visible desde el punto* x_{i+1} *)

$C_{fD_1}^i := C_{fD_1}^i \cup C_k$

si no (* *la cara* C_k *no es* ε*-visible desde* x_{i+1} *)

si $(T_j \notin N^i)$ **entonces** (* *si el tetraedro* T_j *no pertenece al núcleo, marcamos sus caras con -1* *)

desde $l := 1$ **hasta** 4 **hace**

hallamos la cara $C_l \in T_j$.

si $(M(C_l) \neq -1)$ **entonces**

$M(C_l) := -1$

fin si

fin desde

llamamos al **procedimiento** INICIALIZACIÓN 1 (* *se inicializan las marcas de las caras* *)

llamamos al **procedimiento** CONJUNTO D_1^i (* *el procedimiento comienza de nuevo* *)

si no (* *el valor de* ε *actual es demasiado restrictivo. Lo cambiamos por* ε_N, *que será más pequeño. Así se asegura que el núcleo sea* ε*-visible* *)

$\varepsilon := \varepsilon_N$

llamamos al **procedimiento** INICIALIZACIÓN 2 (* *se inicializan las marcas de las caras* *)

llamamos al **procedimiento** CONJUNTO D_1^i (* *el procedimiento empieza de nuevo* *)

fin si

fin si

fin si

si no (* *si la cara* C_k *está en la frontera del prisma inicial* *)

$C^i_{fD_1} := C^i_{fD_1} \cup C_k$.

fin si

fin si

fin desde

fin desde

$ND_1^i := ND_1'^i$

$ND_1'^i = \varnothing$

fin mientras

salida: $D_1^i, C_{iD_1}^i, C_{fD_1}^i$.

fin procedimiento

Como dijimos anteriormente, la determinación de D_1^i es la fase más importante del programa. Se ha pretendido que la descripción de los algoritmos que intervienen en ella tenga un carácter lo más general posible. Por ello, en dicha descripción, no se han utilizado los vectores de la estructura de datos, en su lugar, se ha trabajado con los conjuntos que tienen asociados.

La salida de CONJUNTO D_1^i nos proporciona los conjuntos D_1^i, $C_{iD_1}^i$ y $C_{fD_1}^i$. A partir de estos conjuntos se pueden determinar los vectores *LT*, *LCI* y *LCF*, así como el número total de elementos, *tt*, *tci* y *tcf*, de que consta cada uno de ellos. En este procedimiento se lleva a cabo también la determinación de los punteros *ite* y *tite*, a los que no nos hemos referido anteriormente para no complicar en esceso el entendimiento de dicho procedimiento. Supongamos que *nc* es el número global de una cara de $C_{fD_1}^i$ y *j* su número local, entonces, *TCC*(*ite*(*j*),*nc*) nos proporciona el tetraedro exterior a D_1^i que comparte la cara *nc*. Como no siempre existe dicho tetraedro (no existe para las caras del exterior de la triangulación), es necesario saber si ésto ocurre. El puntero *tite*(*j*) es 1 si existe tal tetraedro y 0 en caso contrario.

Por último, en este procedimiento se calcula también el vector *ILCF*. Si *nc* es una arista de $C_{fD_1}^i$, entonces, *ILCF*(*nc*) nos proporciona el número local de la misma.

Ni los punteros anteriores ni este último vector son imprescindibles en la estructura de datos del programa; su finalidad estriba en disminuir la complejidad de algunos procedimientos.

La función de los procedimientos restantes es la de renovar la estructura de datos, tienen por tanto un carácter más particular. Por este motivo se ha preferido que en su descripción aparezcan explícitamente los vectores de la estructura de datos.

El procedimiento anterior asegura que se cumplan los dos primeros requisitos que exigimos a D_1^i; a saber: que un tetraedro T pertenezca a D_1^i sólo si $x_{i+1} \in B(T)$ y que el conjunto D_1^i sea ε-en forma de estrella. Sin embargo, el último requisito: el conjunto D_1^i no debe contener ningún punto de X aparte de x_{i+1}, puede que no se satisfaga. Para que esto sea así, los errores de redondeo en el cálculo de las coordenadas del centro de las esferas circunscritas deben ser comparables a la distancia mínima entre puntos de X. En la figura 19 (a) se muestra de forma gráfica, en dimensión dos, esta posibilidad. El error en el cálculo de los radios (en realidad, los cuadrados de los radios) es prácticamente despreciable ya que el número de operaciones necesarias es muy reducido (tres sumas y tres multiplicaciones).

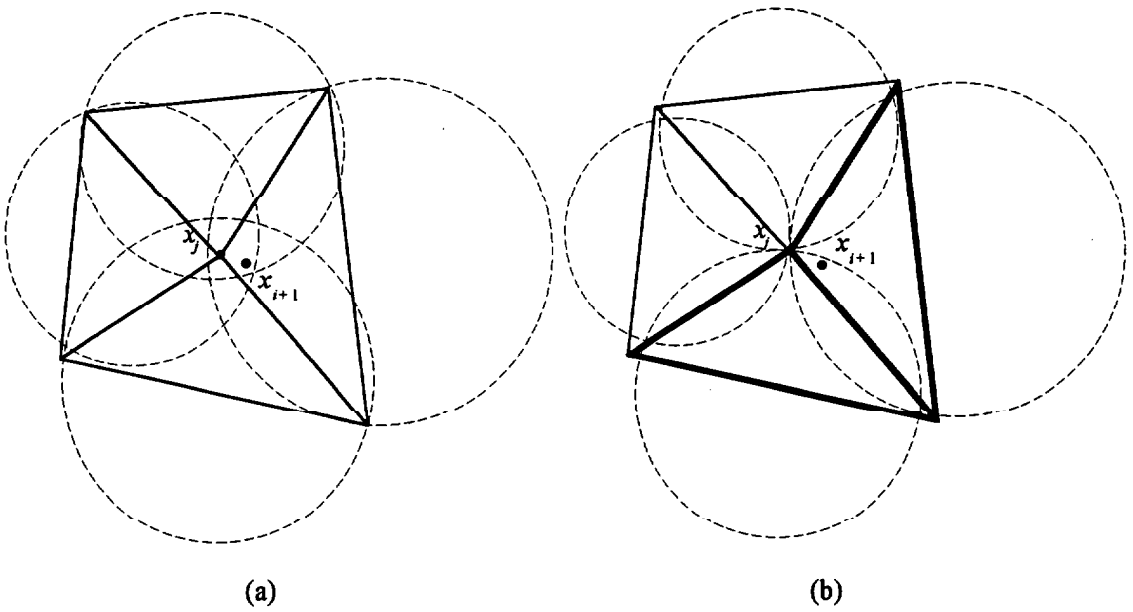


Figura 19. Conjunto D_1^i *conteniendo otro punto aparte de* x_{i+1} (a). *Corrección de la anomalía anterior mediante la reducción de los radios de los círculos* (b).

Supongamos que para un tetraedro T definimos el *radio mínimo* de su esfera circunscrita como:

$$rmin_T = \min_{1 \le j \le 4} \{d(c_T, v_j)\}$$

donde c_T es el centro, con la posible imprecisión que introduce el ordenador, de dicha esfera y v_j es el vértice j-ésimo de T. Si en la determinación de D_1^i utilizamos los *radios mínimos* de los tetraedros, será imposible, a pesar de las imprecisiones, que el conjunto D_1^i contenga otro punto que no sea x_{i+1}.

En la practica, la diferencia entre los valores máximo y mínimo de los radios es muy pequeña. En concreto, la máxima diferencia observada en diversas aplicaciones ha sido $1{,}364 \times 10^{-12}$, y su valor relativo, es decir, el cociente entre esta diferencia y el *radio mínimo*, ha sido $1{,}135 \times 10^{-13}$.

2.5.4. Determinación de *LAF* y *LAI*.

El vector *LAF* es el encargado de almacenar los números globales de las aristas de $A^i_{fD_1}$. El procedimiento LAF actualiza el vector *LAF* mediante un bucle en caras de $C^i_{fD_1}$. Hay que tener en cuenta que cada arista de $A^i_{fD_1}$ pertenece a dos caras de $C^i_{fD_1}$, y por consiguiente, aparece dos veces en el bucle en caras. El procedimiento debe evitar que éstas sean incluidas en *LAF* más de una vez.

procedimiento LAF

entrada: *LCF, tcf, AC*

cont := 0

desde *j* := 1 **hasta** *tcf* **hace**

  nc := *LCF*(*j*)

  desde *k* := 1 **hasta** 3 **hace**

    na := *AC*(*k*, *nc*)

      si (*na aparece por primera vez*) **entonces**

$cont := cont + 1$

$LAF(cont) := na$

fin si

fin desde

fin desde

$taf := cont$

salida: *LAF, taf*

fin procedimiento.

Aunque por razones de simplicidad no se ha indicado, el bucle en $C^i_{f D_1}$ realizado en el procedimiento anterior se utiliza para establecer los punteros *ice*, *tice*, *icf* cuyo significado se explica a continuación. Si *na* es una arista de $A^i_{f D_1}$ con número local *j*, la cara determinada por $CCA(ice(k,j),na)$ es la *k*-ésima cara exterior a D^i_1 que comparte *na* y $tice(j)$ es el número total de éstas. De manera similar, la cara determinada por $CCA(icf(k,j),na)$ es la *k*-ésima cara de $C^i_{f D_1}$ que comparte *na*; obviamente, el número total de éstas es siempre dos.

Otro vector que se calcula en el procedimiento anterior y al que tampoco se a hecho referencia es *ILAF*. Así, si *na* es una arista de $A^i_{f D_1}$, entonces, $ILAF(na)$ nos proporciona el número local de la misma.

Tampoco en esta ocasión los punteros y el vector aquí señalados son imprescindibles para el funcionamiento del programa; su única función es acelerar los procedimientos donde intervienen.

Los números globales de las aristas que desaparecen en cada iteración, las del conjunto $A^i_{i D_1}$, se almacenan en el vector *LAI*. El procedimiento LAI actualiza el vector *LAI* mediante un bucle en caras de $C^i_{i D_1}$. En esta ocasión cada arista de $A^i_{i D_1}$ aparece dos o más veces en el bucle en caras. Este hecho debe ser tenido en cuenta por el

procedimiento. Algunas de las aristas de las caras de $C^i_{iD_1}$ pueden pertenecer a $A^i_{fD_1}$, y por tanto, no deben ser incluidas en *LAI*.

procedimiento LAI

entrada: *nra, nrc, LCI, tci, AC, LAF*

$cont := nra$

desde $j := nrc + 1$ **hasta** *tci* **hace**

$nc := LCI(j)$

desde $k := 1$ **hasta** 3 **hace**

$na := AC(k, nc)$

si $(na \notin LAF)$ **entonces**

si (*na aparece por primera vez*) **entonces**

$cont := cont + 1$

$LAI(cont) := na$

fin si

fin si

fin desde

fin desde

$tai := cont$

salida: *LAI, tai*

fin procedimiento

2.5.5. Determinación de *LVF*.

Los números globales de los vértices de $V^i_{fD_1}$ se almacenan en el vector *LVF*. El procedimiento LVF actualiza el contenido de dicho vector mediante un bucle en artistas

de $A^i_{fD_1}$. Un vértice de $V^i_{fD_1}$ pertenece a varias aristas de $A^i_{fD_1}$, por consiguiente, hay que evitar que éste aparezca repetido en *LVF*.

procedimiento LVF

entrada: *LAF*, *taf*, *VA*

cont := 0

desde *j* := 1 **hasta** *taf* **hace**

na := *LAF*(*j*)

desde *k* := 1 **hasta** 2 **hace**

nv := *VA*(*k*, *na*)

si (*nv aparece por primera vez*) **entonces**

cont := *cont* + 1

LVF(*cont*) := *nv*

fin si

fin desde

fin desde

tvf := *cont*

salida: *LVF*, *tvf*

fin procedimiento

En esta ocasión el bucle en $A^i_{fD_1}$ se aprovecha para calcular los punteros *iae*, *tiae*, *iaf* y *tiaf*. Si *nv* es el número global de un vértice de $V^i_{fD_1}$ y *j* es su número local, entonces, la arista determinada por *ACV*(*iae*(*k*, *j*),*nv*) es la *k*-ésima arista exterior a D^i_j que comparte el vértice *nv* y *tiae*(*j*) el número total de ellas. Por su parte, *ACV*(*iaf*(*k*, *j*),*nv*) es la *k*-ésima arista de $A^i_{fD_1}$ que comparte *nv* y *tiaf*(*j*) su total.

2.5.6. Actualización de *VT*.

El vector *VT* almacena los números globales de los vértices de los tetraedros. Cada vez que se introduce un nuevo punto es necesario actualizar *VT*. Dado que los tetraedros de D_1^i van a desaparecer, la numeración de tetraedros en el vector *VT* debe retomar los números almacenados en *LT* (los residuales más los que desaparecen) y números superiores a *ntt* cuando éstos se agoten. Como cada cara de $C_{f\,D_1}^i$ determina un nuevo tetraedro, el procedimiento VT tiene que realizar un bucle en caras de este conjunto.

procedimiento VT

entrada: *LCF, tcf, LT, tt, VC, ntt, ntv*
desde $j := 1$ **hasta** *tcf* **hace**
  $nc := LCF(j)$
  si $(j \le tt)$ **entonces**
    $nt := LT(j)$
  si no
    $ntt := ntt + 1$
    $nt := ntt$
  fin si
  $LT2(j) := nt$
  $VT(1,nt) := ntv + 1$
  desde $k := 1$ **hasta** 3 **hace**
    $k' := k'(k)$ (* *k' es una permutación de k de manera que la ordenación de los vértices del tetraedro sea la correcta* *)
  $VT(k+1,nt) := VC(k',nc)$
fin desde

fin desde

si ($tcf < tt$) **entonces** (* *existen tetraedros residuales* *)

desde $j := tcf + 1$ **hasta** tt **hace** (* *anotamos los números de los tetraedros residuales* *)

$LT(j - tcf) := LT(j)$

fin desde

$nrt := tt - tcf$

si no

$nrt := 0$

fin si

salida: *VT, LT, nrt, LT2*

fin procedimiento

2.5.7. Actualización de *VC*.

El vector *VC* almacena los números globales de los vértices de las caras. El proceso para actualizar este vector es similar al descrito anteriormente, salvo que ahora el bucle se realiza en aristas de $A^i_{fD_1}$.

procedimiento VC

entrada: *LAF, taf, LCI, tci, VA, ntc, ntv*

desde $j := 1$ **hasta** *taf* **hace**

$na := LAF(j)$

si ($j \leq tci$) **entonces**

$nc := LCI(j)$

si no

$ntc := ntc + 1$

$nc := ntc$

fin si

$LC2(j) := nc$

$VC(1,nc) := ntv + 1$

desde $k := 1$ **hasta** 2 **hace**

$VC(k+1,nc) := VA(k,na)$

fin desde

fin desde

si ($taf < tci$) **entonces** (* *existen caras residuales* *)

desde $j := taf + 1$ **hasta** tci **hace** (* *anotamos los números de las caras residuales* *)

$LCI(j - taf) := LCI(j)$

fin desde

$nrc := tci - taf$

si no

$nrc := 0$

fin si

salida: *VC, LCI, nrc, LC2*

fin procedimiento

2.5.8. Actualización de *VA*.

El vector *VA* almacena los números globales de los vértices de las aristas. De nuevo, el proceso que se sigue para actualizar este vector es similar al descrito en los apartados anteriores salvo que ahora el bucle se realiza en vértices de $V^i_{fD_1}$.

procedimiento VA

entrada: *LVF, tvf, LAI, tai, nta, ntv*

desde $j := 1$ **hasta** tvf **hace**

$nv := LVF(j)$

si $(j \leq tci)$ **entonces**

$na := LAI(j)$

si no

$nta := nta + 1$

$na := nta$

fin si

$LA2(j) := na$

$VA(1, na) := ntv + 1$

$VA(2, na) := nv$

fin desde

si $(tvf < tai)$ **entonces** (* *existen aristas residuales* *)

desde $j := tvf + 1$ **hasta** tai **hace** (* *anotamos los números de las aristas residuales* *)

$LAI(j - tvf) := LAI(j)$

fin desde

$nra := tai - tvf$

si no

$nra := 0$

fin si

salida: VA, LAI, nra, $LA2$

fin procedimiento

2.5.9. Actualización de *CT* y *TCC*.

Esta parte del programa es la encargada de actualizar los vectores *CT* y *TCC*. El vector *CT* almacena los números globales de las caras de cada tetraedro. El vector *TCC* determina los números globales de los tetraedros que comparten cada cara. Básicamente el proceso se descompone en dos partes. La primera realiza un bucle en caras de $C^i_{fD_1}$ para actualizar los vectores anteriores en lo relativo a estas caras. La segunda hace lo propio con las caras de $C^i_{iD_1}$. Para ello realiza un bucle en aristas de $A^i_{fD_1}$, cada una de las cuales define una cara de $C^i_{iD_1}$.

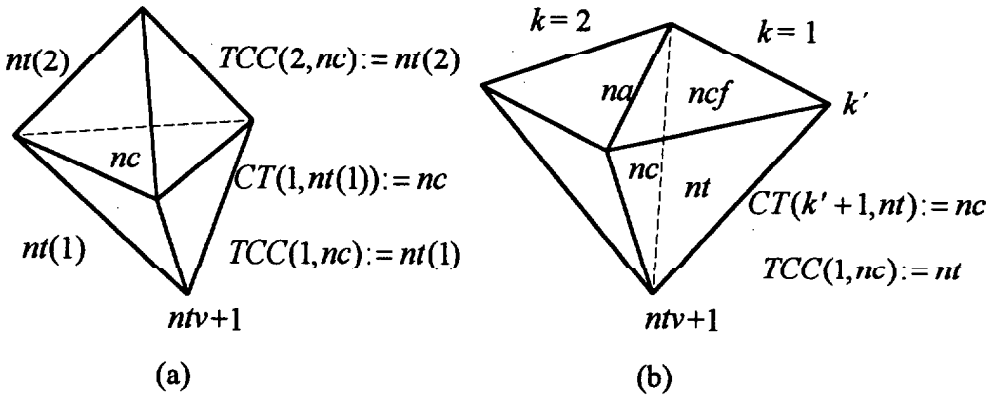


Figura 20. Ilustración del procedimiento CTTCC

En la figura 20 (a) se muestra la primera parte del procedimiento CTTCC, y en la 19 (b), la segunda.

procedimiento CTTCC

entrada: *CT, TCC, ttcc, LCF, tcf, LAF, taf, ite, tite, icf, LT2, LC2, ILCF*

desde $j:=1$ **hasta** *tcf* **hace** (* *Parte primera: actualización de los datos relativos a las caras de* $C^i_{fD_1}$ *)

nc:= *LCF*(*j*) (* *j es el nº local de la cara nc y del tetraedro construido a partir de ella* *)

$nt(1) := LT2(j)$ (* *el tetraedro* $nt(1)$ *perteneciente a* D_2^i *)

$CT(1,nt(1)) := nc$

$cont := 1$

desde $k := 1$ **hasta** $tite(j)$ **hace**

$cont := cont + 1$

$nt(cont) := TCC(ite(j),nc)$

fin desde

$ttcc(nc) := cont$

desde $k := 1$ **hasta** $ttcc(nc)$ **hace** (* *ordenación de TCC para la cara nc* *)

$TCC(k,nc) := nt(k)$

fin desde

fin desde

desde $j := 1$ **hasta** taf (* *Parte segunda: actualización de los datos relativos a las caras de* $C_{iD_1}^i$ *)

$na := LAF(j)$ (* *j es el nº local de la arista na y de la cara nc construida a partir de ella* *)

$nc := LC2(j)$ (* *la cara nc perteneciente a* C_2^i *)

desde $k := 1$ **hasta** 2 **hace**

$ncf := CCA(icf(k,j),na)$ (* *la cara ncf pertenece a* $C_{fD_1}^i$ *)

$nlc := ILCF(ncf)$

$nt := LT2(nlc)$ (* *el tetraedro nt perteneciente a* D_2^i, *es el construido a partir de la cara ncf* *)

$TCC(k,nc) := nt$

hallamos el número k' *tal que el vértice* $VC(k',ncf)$ *sea el opuesto a los vértices de la arista na*

$CT(k'+1,nt) := nc$

fin desde

$ttcc(nc) := 2$

fin desde

salida: *CT, TCC, ttcc*

fin procedimiento

2.5.10. Actualización de *AC* y *CCA*.

Siguiendo un esquema similar al descrito en el procedimiento anterior actualizaremos los vectores *AC* y *CCA*. El vector *AC* almacena los números globales de las aristas de cada cara mientras que el vector *CCA* establece cuales son los números globales de las caras que comparten cada arista. También en este procedimiento se distinguen dos partes fundamentales. La primera actualiza los vectores anteriores para las aristas de $A^i_{fD_1}$, y la segunda lo hace para las aristas de $A^i_{iD_1}$.

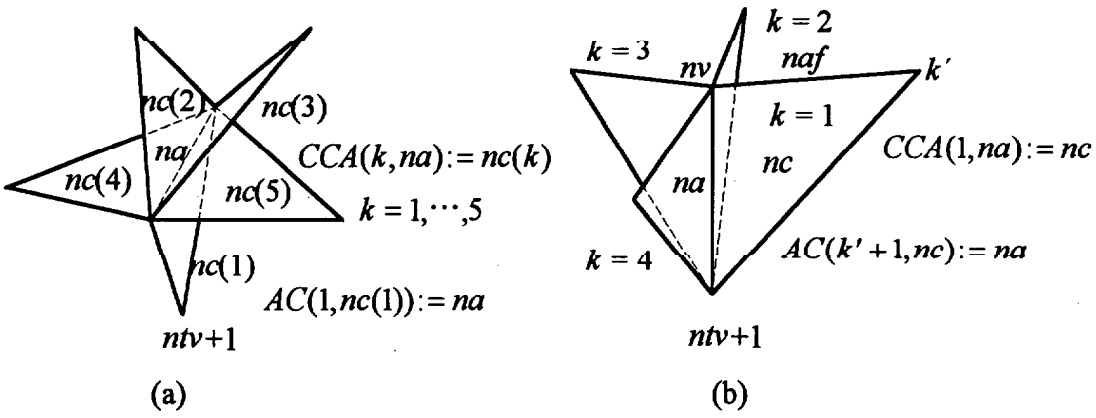


Figura 21. Ilustración del procedimiento ACCCA.

En la figura 21 se muestran las dos partes del proceso seguido por ACCA.

procedimiento ACCCA

entrada: *AC, CCA, tcca, LAF, taf, LVF, tvf, ice, tice, icf, iaf, tiaf, LC2, LA2, ILAF*

desde $j:=1$ **hasta** *taf* **hace** (* *Primera parte: actualización de los datos relativos a las aristas de* $A^i_{fD_1}$ *)

$na:=LAF(j)$ (* *j es el nº local de la arista na y de la cara construida a partir de ella* *)

$nc(1):=LC2(j)$ (* *la cara* $nc(1)$ *perteneciente a* C^i_2 *)

$AC(1,nc(1)):=na$

$cont:=1$

desde $k:=1$ **hasta** $tice(j)$ **hace**

$cont:=cont+1$

$nc(cont):=CCA(ice(k,j),na)$

fin desde

desde $k:=1$ **hasta** 2 **hace**

$cont:=cont+1$

$nc(cont):=CCA(icf(k,j),na)$

fin desde

$tcca(na):=cont$

desde $k:=1$ **hasta** $tcca(na)$ **hace** (* *ordenación de CCA para la arista na* *)

$CCA(k,na):=nc(k)$

fin desde

fin desde

desde $j:=1$ **hasta** *tvf* (* *Segunda parte: actualización de los datos relativos a las caras de* $C^i_{iD_1}$ *)

$nv:=LVF(j)$ (* *j es el nº local del vértice nv y de la arista na construida a partir de él* *)

$na:=LA2(j)$ (* *la arista na perteneciente a* A^i_2 *)

desde $k:=1$ **hasta** $tiaf(j)$ **hace**

$naf:=ACV(iaf(k,j),nv)$ (* *la arista naf pertenece a* $A^i_{fD_1}$ *)

$nla:=ILAF(naf)$

nc:= *LC2*(*nla*) (* *la cara nc perteneciente a* C_2^i, *es la construida a partir de la arista naf* *)

CCA(*k*,*na*):= *nc*

hallamos el número k' tal que el vértice VA(*k'*,*naf*) *sea el opuesto al vértice nv*

AC(*k'* + 1,*nc*):= *na*

fin desde

tcca(*na*):= *tiaf* (*j*)

fin desde

salida: *AC, CCA, tcca*

fin procedimiento

2.5.11. Actualización de *ACV*.

El proceso de inclusión del punto *ntv*+1 concluye con la actualización del vector *ACV*. Este vector es el encargado de establecer cuáles son las aristas compartidas por cada vértice.

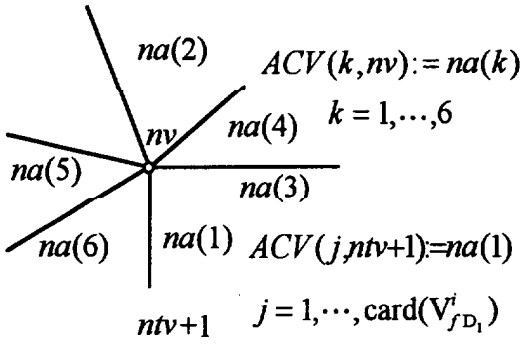


Figura 22. Ilustración del procedimiento ACV.

procedimiento ACV

entrada: *ACV, tacv, LVF, tvf, iae, tiae, iaf, tiaf, LA2*

desde $j := 1$ **hasta** *tvf* **hace** (* *actualización de ACV para los vértices de* $V^i_{fD_1}$ *)

$nv := LVF(j)$ (* *j es el nº local del vértice nv y de la arista construida a partir de él* *)

$na(1) := LA2(j)$ (* *la arista* $na(1)$ *perteneciente a* A^i_2 *)

$cont := 1$

desde $k := 1$ **hasta** $tiae(j)$ **hace**

$cont := cont + 1$

$na(cont) := ACV(iae(k, j), nv)$

fin desde

desde $k := 1$ **hasta** $tiaf(j)$ **hace**

$cont := cont + 1$

$na(cont) := ACV(iaf(k, j), nv)$

fin desde

$tacv(nv) := cont$

desde $k := 1$ **hasta** $tacv(nv)$ **hace** (* *ordenación de ACV para el vértice nv* *)

$ACV(k, nv) := na(k)$

fin desde

$ACV(j, ntv+1) := na(1)$ (* *determinación de ACV para el nuevo vértice* *)

fin desde

$tacv(ntv + 1) := tvf$

salida: *ACV, tacv*

fin procedimiento

Una vez completado este procedimiento el número total de puntos de que consta la triangulación se ha aumentado en una unidad. Si *ntv* es menor que card(X), el

programa considera $ntv := ntv + 1$ y empieza de nuevo la llamada a la secuencia de procedimientos anteriores. En caso contrario, el proceso de triangulación concluye.

El tipo de datos que se necesiten a la salida del programa dependerá de la aplicación que se le vaya a dar a la malla. En todo caso, hay que tener en cuenta que la triangulación del dominio es una parte de la triangulación de los puntos de X más los ocho del cubo inicial. Por tanto, será necesario aislar los elementos pertenecientes al dominio de los que no lo son. En el capítulo siguiente se explicará el proceso seguido para esta separación. Aunque es una situación muy infrecuente, puede ocurrir que una vez se hayan triangulado todos los puntos de X, los punteros *nrt*, *nrc* o *nra* sean no nulos. Esto significaría que los vectores LT, LCI o LAI tienen almacenados los números correspondientes a elementos que en realidad no existen en la triangulación, y que por consiguiente, deben ser excluidos de cualquier aplicación que se le de a ésta.

3. GENERACIÓN AUTOMÁTICA DE PUNTOS

3.1. Introducción.

El programa de triangulación descrito en capítulo segundo construye la triangulación de Delaunay de un conjunto de puntos previamente definido. Por tanto, es necesario disponer de un programa que genere puntos automáticamente sobre la superficie y en el interior del dominio a triangular. La distribución de estos puntos tiene que atender a dos cuestiones fundamentales:

- La triangulación debe *conservar la geometría* del dominio.

- La *calidad* de la triangulación resultante debe ser aceptable.

El significado preciso de estos requisitos se dará más adelante.

Sólo los dominios poliédricos pueden ser triangulados exactamente. Una triangulación tan sólo puede representar de forma aproximada dominios que posean superficies curvas. El problema de aproximación de superficies curvas mediante superficies poliédricas, y la consiguiente generación de puntos, son asuntos bastante complejos que no trataremos aquí.

En este capítulo se explican los procedimientos empleados para describir dominios poliédricos y la manera en que se generan puntos sobre ellos.

3.2. Descripción del dominio.

El problema de generar puntos de forma automática en el interior y en la superficie de un dominio se simplifica substancialmente si éste se descompone en una colección de k-poliedros convexos ($k = 0,1,2$ ó 3) "casi" disjuntos. Esto es, k-poliedros cuyas intersecciones dos a dos, o bien sean vacías, o bien estén contenidas en sus fronteras. De esta manera evitamos que exista superposición, o una proximidad excesiva, entre los puntos generados de forma independiente en cada k-poliedro.

Descompondremos el dominio poliédrico Ω (3-poliédrico en concreto) en un conjunto de k-poliedros convexos, P_i^k , ($k = 0,1,2$ ó 3) de manera que Ω sea la unión de todos ellos

$$\Omega = \bigcup_{0 \le k \le 3} \{ \bigcup_i \overset{\circ}{P}_i^k \}$$

y las intersecciones entre ellos verifiquen que

$$\overset{\circ}{P}_i^k \cap \overset{\circ}{P}_j^l = \varnothing \quad (i \neq j \text{ y } k,l = 0,1,2,3)$$

Para abreviar la notación, se ha supuesto que los interiores relativos de los 0-poliedros, es decir, los vértices, son los propios 0-poliedros.

Todos los k-poliedros en que se descompone el dominio están numerados. La descripción de los 0-poliedros (vértices) se hace por medio de sus coordenadas. Los 1-poliedros (aristas) se describen indicando los números de referencia de sus vértices. Los 2-poliedros (caras poligonales) se describen por medio de los números de referencia de sus aristas. De forma general, cada k-poliedro ($k = 1,2$ ó 3) se describe dando los números de referencia de los (k-1)-poliedros que determinan su frontera.

3.3. Generación de puntos.

En cada k-poliedro ($k = 1,2$ ó 3) se generan puntos en su interior con unas distancias entre ellos especificadas en la entrada de datos. Si la distribución de puntos es bastante uniforme, los tetraedros que resulten de la triangulación serán muy regulares, es decir, *casi* equiláteros. Como veremos más adelante, una triangulación tridimensional en la que sus tetraedros sean muy regulares tendrá una *calidad* elevada. Hay varios factores que influyen en la uniformidad de la distribución de puntos. En primer lugar, las densidades de puntos fijadas en la entrada de datos deben ser similares para todos los k-poliedros que estén próximos entre si. En segundo lugar, supongamos que d sea un valor representativo de las distancias entre puntos situados sobre la frontera de un k-poliedro, en este caso $k = 2$ ó 3. La mínima distancia que debe existir entre los puntos generados en el interior del k-poliedro y su frontera debe ser del orden de $d/2$. De esta manera se creará una banda sin puntos, de una anchura próxima a $d/2$, alrededor de la frontera de cada k-poliedro.

Las dos condiciones anteriores no sólo logran que la disposición de puntos sea bastante uniforme, sino que también consiguen, salvo en dominios muy complejos, que la triangulación *conserve la geometría* del dominio. De este asunto nos ocuparemos en el apartado 3.6.

La generación de puntos en cada k-poliedro tiene un tratamiento específico según su dimensión. Por esta razón y por ser expresiones de mayor uso, a partir de ahora utilizaremos los términos, arista, cara poligonal (o simplemente cara) y región poliédrica (o simplemente poliedro) para referirnos a los correspondientes k-poliedros.

3.3.1. Generación de puntos en las aristas.

Supongamos una arista A definida por sus vértices extremos x_i y x_f. La distancia deseada, δ_A, entre los puntos generados en el interior de A se especifica en la entrada de datos. El programa ajusta esta distancia para conseguir que la distancia entre puntos generados sea la misma que la existente entre un punto extremo y su adyacente. De esta manera se consigue mayor regularidad en la triangulación resultante.

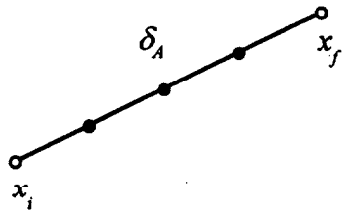


Figura 23. Generación de puntos en la arista A.

3.3.2. Generación de puntos en las caras poligonales.

Consideremos dos aristas de la cara poligonal C que sean no paralelas y confluentes en un mismo punto x_o. Sobre estas aristas se definen dos vectores unitarios $\vec{u}_1$ y $\vec{u}_2$ linealmente independientes. Los puntos interiores de C se generan en las direcciones marcadas por $\vec{u}_1$ y $\vec{u}_2$ guardando unas distancias apropiadas con las aristas que delimitan la cara. En la entrada de datos se especifican las distancias entre puntos generados correspondientes a cada dirección, así como el punto de confluencia x_o. En general, la elección de este punto se hace de manera que el ángulo formado por $\vec{u}_1$ y $\vec{u}_2$ sea los más próximo posible a $\pi/2$.

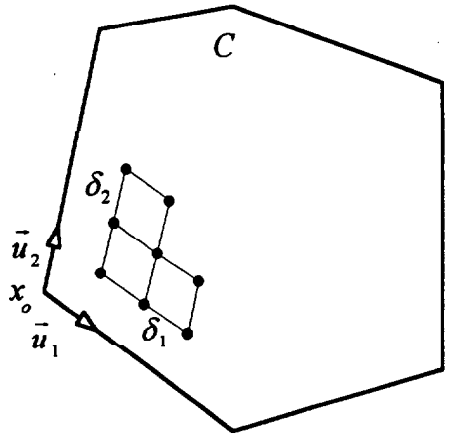


Figura 24. Generación de puntos en la cara C.

En la figura 24 podemos ver que los puntos interiores a C se generan en las direcciones $\vec{u}_1$ y $\vec{u}_2$ con unas distancias entre ellos δ_1 y δ_2, respectivamente.

Sea δ_A la distancia entre puntos en la arista A. Supongamos que x_1 y x_2 sean dos puntos cualesquiera pertenecientes a las caras C_1 y C_2 con arista común A. Si la distancia mínima a la que pueden estar los puntos x_1 y x_2 de la arista A es mayor que $d_A = 1/2\delta_A$, la arista A nunca será atravesada por el segmento $\overline{x_1x_2}$. Esto se puede demostrar teniendo en cuenta que si se formara el segmento $\overline{x_1x_2}$ y éste cortara a A, cualquiera de las circunsferas de $\overline{x_1x_2}$ tendría un radio $r > d_A$, y por tanto, su diámetro sería mayor que δ_A. En consecuencia, cualquiera de estas circunsferas contendría, necesariamente, puntos de A, contradiciendo la propiedad de las esferas vacías (ver figura 25).

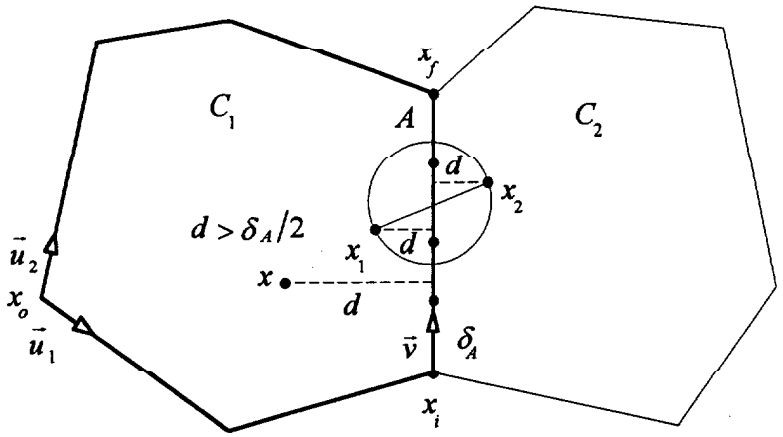


Figura 25. Criterio de distancias entre aristas y puntos interiores a las caras.

La distancia d entre un punto cualquiera de la cara C y la arista A se calcula mediante la expresión:

$$d = \left| (\vec{v} \times \vec{n}) \cdot \overline{xx_i} \right|$$

donde $\vec{n} = \vec{u}_1 \times \vec{u}_2 / \left\| \vec{u}_1 \times \vec{u}_2 \right\|$ es el vector unitario normal a C y $\vec{v} = \overline{x_i x_f} / \left\| \overline{x_i x_f} \right\|$ es el vector unitario en la dirección de la arista A, tal y como se representa en la figura 25 para la cara C_1. Además, la cantidades $(\vec{v} \times \vec{n}) \cdot \overline{xx_i}$ calculadas anteriormente permiten saber cuando un punto x está dentro de una cara C; el punto x es interior a C si y sólo si se verifica que $(\vec{v} \times \vec{n}) \cdot \overline{xx_i} > 0$ para cada una de las aristas de la cara.

Como conclusión, si tomamos $d_A = 1/2\,\delta_A$ como la distancia mínima a la que pueden estar los puntos generados en C de la arista A, la arista no será *destruida* por efecto de dichos puntos.

3.3.3. Generación de puntos en las regiones poliédricas.

En este caso se necesitan tres aristas no coplanarias y confluentes en un mismo punto para definir los vectores unitarios $\vec{u}_1$, $\vec{u}_2$ y $\vec{u}_3$. Los puntos interiores en un poliedro P se generan en las direcciones de $\vec{u}_1$, $\vec{u}_2$ y $\vec{u}_3$ con unas distancias entre ellos δ_1, δ_2 y δ_3 especificadas en la entrada de datos. Al igual que en el caso de las caras poligonales, el punto de confluencia x_o se escoge de manera que los ángulos formados por los vectores unitarios sean lo más próximos posible a $\pi/2$.

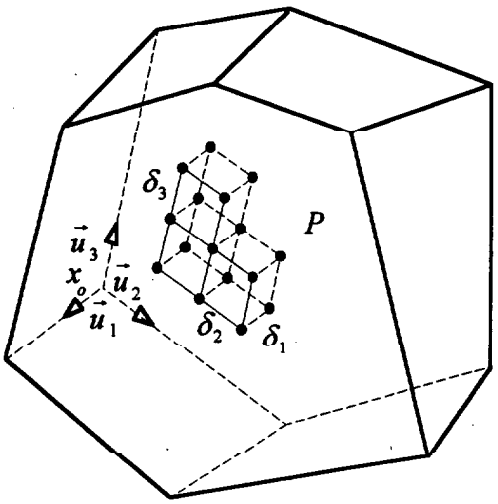


Figura 26. Generación de puntos en el poliedro P.

Para evitar que los puntos generados en el interior de P puedan *destruir* las caras que lo delimitan, será necesario que se mantengan unas distancias mínimas entre dichas caras y los puntos interiores. Ahora el problema es algo más complicado debido a que estas distancias dependen del mayor circulo, o al menos una cota superior, que se puede construir en cada cara sin incluir puntos de su interior ni de su frontera. Este problema es un caso especial del problema clásico (MAYOR CÍRCULO VACÍO): dado un conjunto S de N puntos del plano, encontrar el mayor circulo que no contenga puntos de S en su interior y cuyo centro sea interno a la envolvente convexa de S [PreSha85].

Para encontrar el mayor círculo vacío se construye el diagrama de Voronoi, $Vor(S)$, y la envolvente convexa, conv(S), asociados a S. La intersección de $Vor(S)$ con conv(S) se puede considerar como una colección de poliedros convexos. Se demuestra que el mayor círculo vacío está centrado en algún vértice de uno de estos poliedros. Este vértice es un vértice de $Vor(S)$ o la intersección de una arista de $Vor(S)$ con otra de conv(S).

La regularidad con que están dispuestos los puntos en las caras permite encontrar una cota superior del mayor círculo vacío sin necesidad de construir completamente el diagrama de Voronoi.

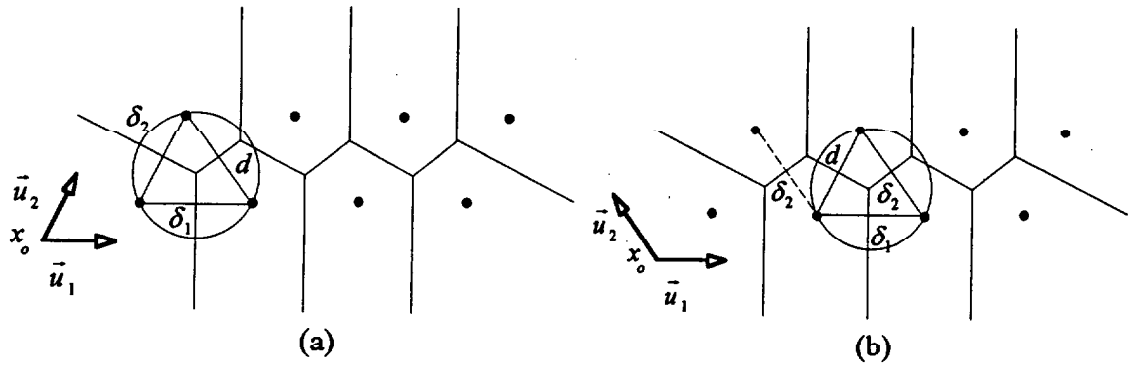


Figura 27. Diagrama de Voronoi y mayor círculo vacío para los puntos interiores a una cara con $\vec{u}_1 \cdot \vec{u}_2 > 0$ (a), *y con* $\vec{u}_1 \cdot \vec{u}_2 < 0$ (b).

El diagrama de Voronoi correspondiente a los puntos interiores a la cara tiene una estructura repetitiva. Los círculos centrados en los vértices del diagrama son los círculos circunscritos a los triángulos de lados δ_1, δ_2 y d. Sus radios siempre tienen el mismo valor, r_i:

$$r_i = \frac{\delta_1 \delta_2 d}{4\sqrt{p(p-\delta_1)(p-\delta_2)(p-d)}}$$

donde δ_1 y δ_2 son las distancias entre puntos en las direcciones de $\vec{u}_1$ y $\vec{u}_2$, $2p = \delta_1 + \delta_2 + d$ es el perímetro del triángulo, y d es uno de sus lados, dado por la expresión $d^2 = \delta_1^2 + \delta_2^2 - 2\delta_1\delta_2\, \vec{u}_1 \cdot \vec{u}_2$ cuando $\vec{u}_1 \cdot \vec{u}_2 > 0$, y por $d^2 = \delta_1^2 + \delta_2^2 + 2\delta_1\delta_2\, \vec{u}_1 \cdot \vec{u}_2$ cuando $\vec{u}_1 \cdot \vec{u}_2 < 0$. Teniendo en cuenta que los triángulos que definen el mayor círculo vacío son siempre acutángulos (ver figura 27), o a lo sumo tienen un ángulo recto, se puede demostrar que se verifica la desigualdad $r_i \le 1/2\sqrt{\delta_1^2 + \delta_2^2}$ (la igualdad corresponde al caso $\vec{u}_1 \cdot \vec{u}_2 = 0$).

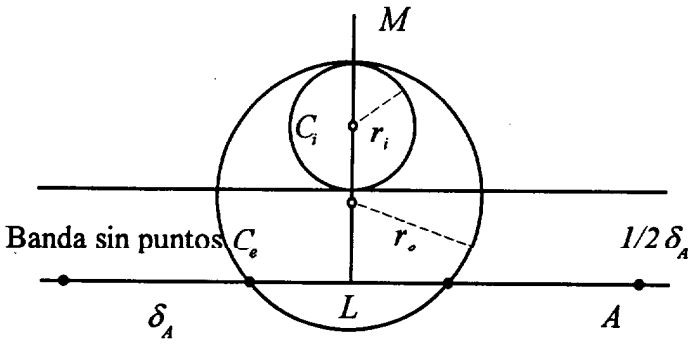


Figura 28. Cota superior al mayor círculo vacío formado con puntos de A.

En las proximidades de las aristas de la cara la estructura del diagrama de Voronoi se complica, haciendo difícil calcular a priori el mayor círculo vacío. En cambio, lo que sí resulta sencillo es establecer una cota superior a este círculo.

Los vértices del diagrama de Voronoi en las proximidades de la arista A están situados en puntos de las mediatrices de los segmentos en que se divide A (ver figura 28). Sea L uno de estos segmentos y M su mediatriz. Los mayores círculos que se puede formar sin contener puntos interiores a la cara tienen radio r_i. Sea C_i uno de estos círculos, que supondremos centrado en M y tangente a la banda sin puntos paralela a A y de anchura $1/2\delta_A$. (cualquier otra situación de este círculo produciría una cota superior menos ajustada). El círculo C_e que pasa por los vértices de L y que contiene a C_i es una cota superior para el mayor círculo construido sobre A y que no contiene puntos de la cara. De hecho, si suponemos que existe un círculo vacío C_e' mayor que C_e, entonces también existirá otro círculo C_i' mayor que C_i y totalmente contenido en C_e'. Como C_i era el mayor círculo vacío que se puede construir en el interior de la cara, C_i' debe contener, forzosamente, puntos interiores de la misma, y en consecuencia, también lo hará C_e', en contradicción con la suposición inicial. El radio de C_e es

$$r_e = \frac{8r_i^2 + \delta_A^2 + 4r_i\delta_A}{2(4r_i + \delta_A)}$$

donde δ_A es la distancia entre puntos en la arista A.

Supongamos que x_1 y x_2 sean dos puntos cualesquiera pertenecientes a los poliedros P_1 y P_2 con cara común C. Si la distancia mínima d_C a la que están los puntos x_1 y x_2 de la cara C es mayor que el valor máximo de r_e para cada una de las aristas de C, entonces, la cara C no será atravesada por el segmento $\overline{x_1x_2}$. Teniendo en cuenta que $r_i \le 1/2\sqrt{\delta_1^2 + \delta_2^2}$ y que $r_e < r_i + 1/2\delta_A$ podemos establecer la siguiente cota de expresión más sencilla para el valor de d_C:

$$d_C = \frac{1}{2}\sqrt{\delta_1^2 + \delta_2^2} + \frac{1}{2}\max\{\delta_{A_i}\},$$

donde δ_{A_i} es la distancia entre puntos en cada una de las aristas de C. Podemos razonar de manera similar a como se hizo anteriormente. Si se formara el segmento $\overline{x_1 x_2}$ y éste cortara a C, cualquiera de las circunsferas de $\overline{x_1 x_2}$ tendría un radio $r > d_C$. Las intersecciones de estas circunsferas con el plano en el que está C serían círculos de radio mayor que d_C y por consiguiente contendrían algún punto de la cara (del interior o de la frontera) violando la propiedad de las esferas vacías.

La distancia d entre un punto x y la cara C perteneciente al poliedro P se calcula mediante la expresión: $d = \left|\overline{xx_v} \cdot \vec{n}\right|$ donde $\vec{n}$ es el vector normal a C, unitario y saliente de P, y x_v es cualquier vértice de C. El punto x está dentro de P si y sólo si $\overline{xx_v} \cdot \vec{n} > 0, \forall C \in P$.

3.4. Ajuste de las densidades de puntos generados.

El procedimiento anterior no asegura que se *respete* la geometría del dominio. En este sentido, lo único que se consigue es que los puntos interiores a una cara o a un poliedro no *destruyan* sus fronteras, pero eso no basta para asegurar la conservación del dominio. Por ejemplo, puede ocurrir que una cara, C, en la que la densidad de puntos sea baja, sea atravesada por segmentos con vértices en otras dos caras situadas a ambos lados de C. Si la forma del dominio no es muy complicada resulta fácil establecer a priori unas densidades de puntos, en las aristas y caras, suficientemente altas como para que no ocurran situaciones como la comentada anteriormente. No obstante, habrá ocasiones en las que esto no sea así. En el apartado 3.6 trataremos este aspecto en más detalle.

En las tres figuras siguientes se muestra la triangulación de un dominio que contiene dos planos y tres líneas interiores cruzándose entre si. Para conseguir la conformidad entre la triangulación y el dominio fue necesario realizar un ajuste de las densidades de puntos en diversas zonas de estos elementos.

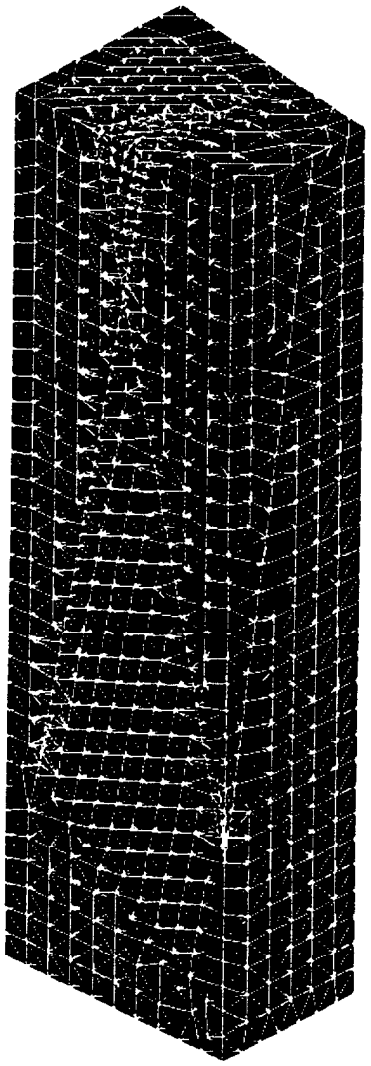


Figura 29. Triangulación del dominio.

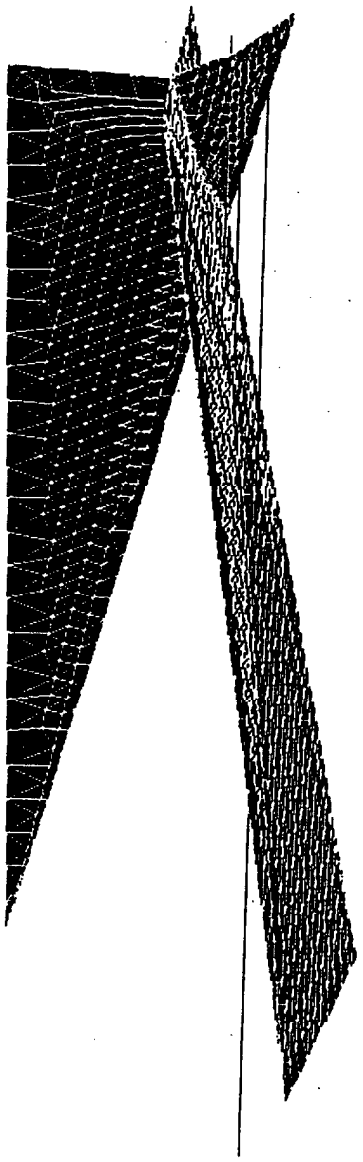


Figura 30. Planos y líneas interiores al dominio.

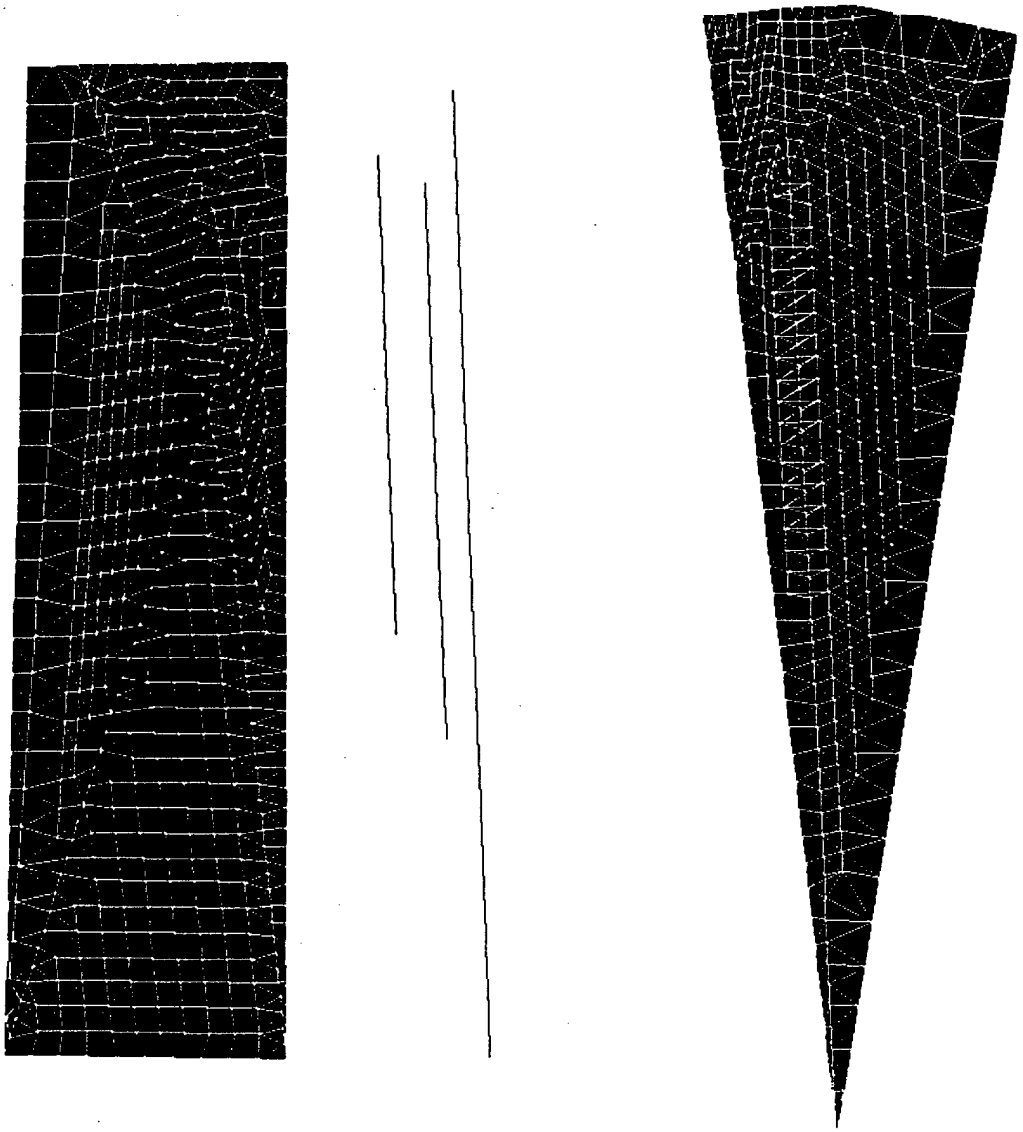


Figura 31. Representación por separado de las líneas y planos interiores.

3.5. Entrada de datos del programa de generación automática de puntos.

En líneas generales, la entrada de datos del programa de generación automática de puntos consiste en una tabla en la que se indica el número total de vértices, aristas, caras poligonales y regiones poliédricas en que se ha descompuesto el dominio. A continuación se indica el número de referencia de cada vértice y sus coordenadas. Los datos que determinan una arista son: su numero de referencia, los números de referencia de sus vértices y la distancia entre puntos a generar en dicha arista. Para determinar las caras damos su número de referencia y el de sus aristas. Además, indicamos el origen del sistema de referencia y la distancia entre puntos en cada una de las direcciones definidas por las aristas que concurren en dicho origen. Las regiones poliédricas se determinan por los números de referencia de sus caras, el origen del sistema de referencia y las distancias entre puntos asociadas a las direcciones definidas por las aristas que confluyen en el origen.

En ocasiones es complicado descomponer el dominio en una colección de *k*-poliedros convexos. Resulta simplificador poder incluir aristas, caras, etc. dentro del dominio sin necesidad de efectuar esta descomposición. La entrada de datos permite considerar algunos elementos como *interiores* al dominio, haciendo que los puntos que los definen estén a distancias apropiadas del resto de puntos. De esta manera se evita que haya puntos superpuestos o excesivamente próximos.

Cada *k*-poliedro de la entrada de datos tiene asociado un número de referencia para indicar alguna propiedad de carácter fisico. Este número es *heredado* por todos los puntos que se generan en el interior relativo del *k*-poliedro.

La salida de datos del programa de generación de puntos es un fichero con una lista en la que se indica el total de puntos que han sido generados, las coordenadas de

dichos puntos acompañadas de su número de orden y su número de referencia de carácter físico.

3.6. Conformidad entre la triangulación y el dominio.

3.6.1. Introducción.

Supongamos que Ω sea un dominio poliédrico. La frontera de Ω está formada por un conjunto de aristas y caras que son definidas en la entrada de datos. Diremos que la triangulación, $T(\Omega)$, es *conforme con la frontera* de Ω, (no confundir con el término introducido en los preliminares: *malla conforme*), si cada una de las aristas y caras previamente definidas es la unión de aristas y caras de $T(\Omega)$.

Básicamente existen dos alternativas para construir una triangulación de Delaunay conforme con la frontera de Ω. La primera consiste en definir (o generar) una triangulación de la superficie de Ω y forzar a que la triangulación resultante contenga todas las caras y aristas de la frontera de Ω, aún cuando la triangulación resultante no sea estrictamente de Delaunay [GeoHec90], [GeoHec91a], [BerEpp92], [WeaHas94], [WrigJac94]. Esta triangulación se conoce como *constrained Delaunay triangulation*. La segunda alternativa consiste en definir los puntos de manera apropiada para que triangulación de Delaunay resultante sea conforme con la frontera de Ω. En dimensión dos existen algoritmos capaces de encontrar, a priori, un conjunto de puntos que cumpla con este requisito [BerEpp92], sin embargo, en dimensión tres no conocemos un algoritmo equivalente

El procedimiento que hemos empleado para tratar de solucionar el problema de la conformidad consiste en:

- Crear un conjunto inicial de puntos con los que se consiga un representación lo más exacta posible del dominio.

- Analizar de forma automática si existe conformidad con la frontera del dominio; en caso de que no la halla se definen nuevos puntos sobre la frontera de Ω en posiciones

adecuadas y se vuelve a triangular. Este proceso se repite hasta que se alcance dicha conformidad.

Por el momento no hemos hecho sino empezar esta línea de trabajo por lo que los resultados obtenidos no son, en absoluto, concluyentes.

3.6.2. Análisis y correción de la conformidad entre la triangulación y la frontera del dominio.

Existen dos situaciones que dan lugar a que no haya conformidad: cuando se produce una intersección entre una de las aristas que definen al dominio y una cara de la triangulación, y cuando se produce una intersección de una de las caras poligonales del dominio con una arista de la triangulación. La intersección de una arista de la triangulación con otra del dominio se puede contemplar en alguno de los casos anteriores si se considera que las intersecciones pueden ocurrir también en las fronteras de las caras.

Intersecciones entre aristas del dominio y caras de la triangulación.

Para detectar estas intersecciones se hace un bucle en aristas del dominio. Para cada una de estas aristas hacemos otro bucle en caras de la triangulación. El conocimiento de los vértices de la arista y de la cara permite determinar, como se verá a continuación, si hay intersección entre estos dos elementos.

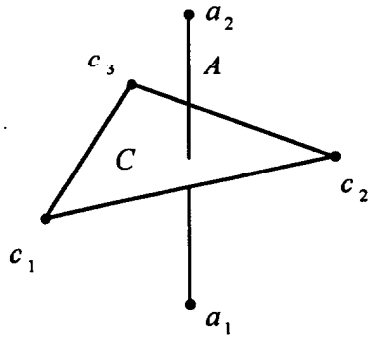


Figura 32. Intersección de una arista del dominio con una cara de la triangulación.

En primer lugar se resuelve el sistema de ecuaciones

$$\alpha\overline{c_1c_2} + \beta\overline{c_1c_3} = \overline{c_1a_1} + \gamma\overline{a_1a_2}$$

donde α, β y γ son las incógnitas, y los diversos vectores que aparecen en la ecuación están construidos a partir de los puntos de la cara y la arista (ver figura 32). Para que la intersección esté entre los límites de la arista A se debe cumplir, supuesto que el sistema sea compatible y determinado, que $0 < \gamma < 1$. Las condiciones para que la intersección esté dentro de la cara son: $0 \le \alpha$, $0 \le \beta$ y $\alpha + \beta \le 1$. La situación en que la arista corta a la cara en uno de sus vértices debe quedar excluido, por tanto, si $\alpha = 0$ ó $\beta = 0$ debe ocurrir que $0 < \beta < 1$ ó $0 < \alpha < 1$, respectivamente.

Intersecciones entre aristas de la triangulación y caras del dominio.

La otra posible causa por la que no se respete la geometría del dominio es porque se produzca una intersección entre una de las aristas de la triangulación y una de las caras poligonales que definen el dominio. Para detectar estas intersecciones se hace un barrido en caras del dominio. Para cada una de estas caras hacemos un barrido en aristas de la triangulación.

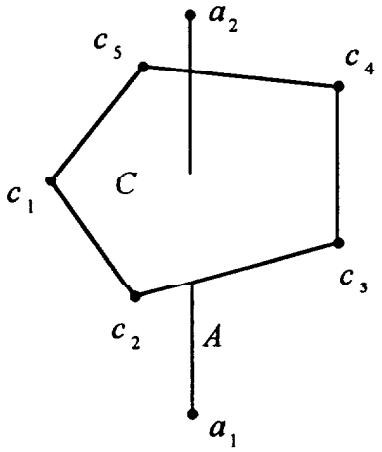


Figura 33. Intersección entre una arista de la triangulación y una cara del dominio.

Para ver si hay intersección entre la arista y el plano que soporta a la cara resolvemos el sistema de ecuaciones

$$\alpha \vec{u}_1 + \beta \vec{u}_2 = \overline{x_o a_1} + \gamma \overline{a_1 a_2}$$

donde $\vec{u}_1$ y $\vec{u}_2$ son los dos vectores unitarios utilizados en la generación de puntos en esta cara y x_o el origen del sistema de referencia. Si el sistema es compatible determinado y $0 < \gamma < 1$ existe intersección entre la arista y el plano que soporta a la cara. Aunque existen algoritmos más rápidos [PreSha85], es posible determinar si la intersección está o no dentro de los límites de la cara mediante un procedimiento igual al utilizado en el apartado 3.3.2. □

En cualquiera de los casos anteriores el cálculo del punto de intersección depende de la solución de un sistema de ecuaciones, o del signo de un determinante, planteados en términos de números reales (números en coma flotante en el ordenador). Está claro que esta forma de proceder presenta problemas de precisión. Por ejemplo, ocurre con frecuencia que los puntos de una cara y de una arista coinciden (ver figura 34). Cualquier pequeño error de redondeo en los cálculos efectuados por el ordenador hace que sea imposible determinar la verdadera localización de la intersección. Es decir, no sabremos si la arista corta a la cara realmente o si forma parte de ella.

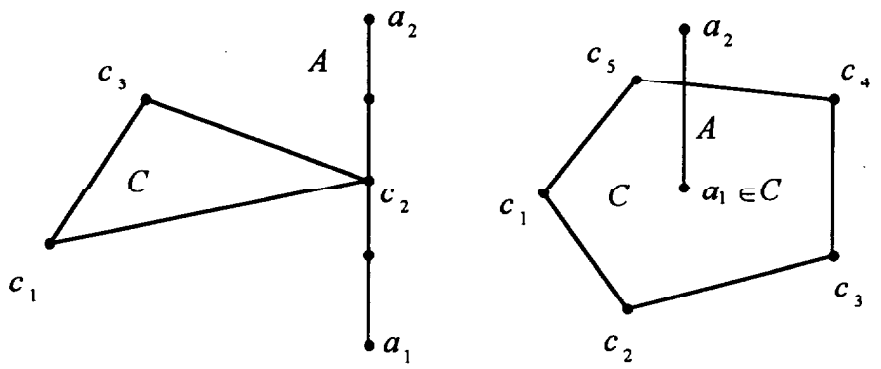


Figura 34. Intersecciones difíciles de clasificar debido a imprecisiones en los cálculos.

Para resolver el problema planteado anteriormente hemos dado a cada punto dos números de referencia mediante la matriz $nrp(i,np)$, donde $i = 1$ ó 2 y np es el número del punto. El primero indica si el punto np es un vértice, $nrp(1,np) = 0$, o si pertenece a una arista, $nrp(1,np) = 1$, a una cara, $nrp(1,np) = 2$, o a una región poliédrica, $nrp(1,np) = 3$, del dominio. El segundo número, $nrp(2,np)$, nos proporciona el número del vértice, la arista, la cara o la región que ha generado este punto. Estos números de referencia nos permiten descartar las situaciones en que una cara y una arista comparten puntos. A pesar de esto, subsisten otros problemas no resueltos por completo hasta el momento, tales como precisar cuando la intersección de una arista ocurre en la frontera de una cara (otra arista) y no en su interior, etc.

La estrategia para conseguir la conformidad con la frontera del dominio consiste en incluir en la triangulación los puntos que definen las intersecciones. Estos puntos *heredan* el número de referencia del elemento del dominio que tratan de recuperar. Existen diferentes posibilidades en cuanto a la secuencia de inclusión de puntos y triangulaciones sucesivas: calcular previamente todas las intersecciones y después proseguir la triangulación, calcular primero las intersecciones de las aristas del dominio con las caras de la triangulación, triangular, después calcular las intersecciones de las caras del dominio con las aristas de la triangulación, y volver a triangular, etc. En cualquiera de los casos se repite el proceso hasta que no aparece ninguna intersección, momento en que se ha conseguido la conformidad. Posiblemente, la mejor opción sea triangular cada vez que se encuentra una nueva intersección. Pero para evitar que el proceso sea muy lento hay que encontrar algún algoritmo que permita no tener que realizar un bucle en todas las caras y aristas cada vez que se añade un nuevo punto. Hemos probado las dos primeras alternativas, siendo la segunda la que ha dado mejores resultados ya que necesita de menos puntos para alcanzar la conformidad, y además, éstos están más distantes entre si.

Aunque parece lógico que cualquiera de estos procesos concluya en un número finito de iteraciones, y además así lo hemos constatado en varias aplicaciones, no tenemos conocimiento de ninguna prueba teórica de este supuesto.

Cada vez que se va ha añadir un nuevo punto se comprueba si está *suficientemente* alejado de los demás; en caso contrario no se incluye en la triangulación. Por supuesto que esta forma de proceder no es sino tan sólo una manera de salir del paso a falta de opciones mejores.

Uno de los problemas que plantea el introducir puntos en las intersecciones es la aparición de tetraedros muy degenerados, los formados con puntos muy cercanos. Ésto se puede evitar en parte si las densidades de puntos se ajustan previamente en función de la complejidad que presente la geometría del dominio en cada zona.

Una característica de este método es que la triangulación final es de Delaunay. Ésta es una diferencia importante frente a otros métodos basados en el intercambio de diagonales y caras en los tetraedros que interceptan alguno de los elementos de la geometría del dominio [GeoHec91], [Weath92], [WeaHas94]. Además, en caso de conseguir buenos resultados, permitiría el refinamiento local de la triangulación incluso en las zonas próximas a la frontera del dominio, sin más que añadir puntos en las zonas en las que las estimaciones del error del MEF sean mayores.

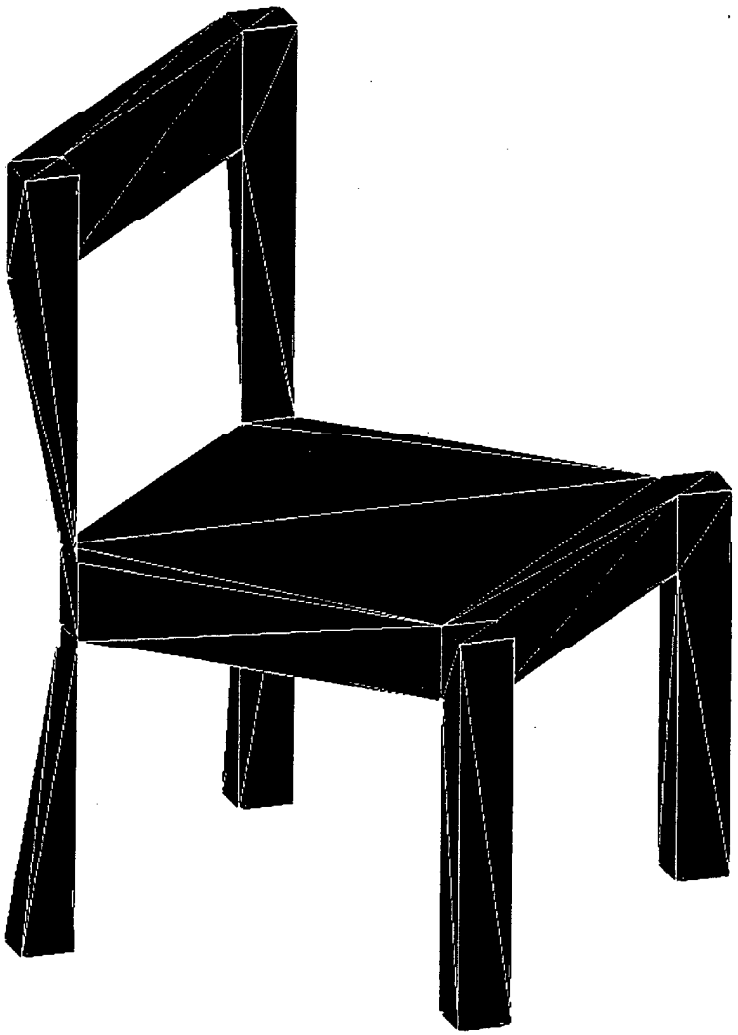


Figura 35. Triangulación no conforme con el dominio.

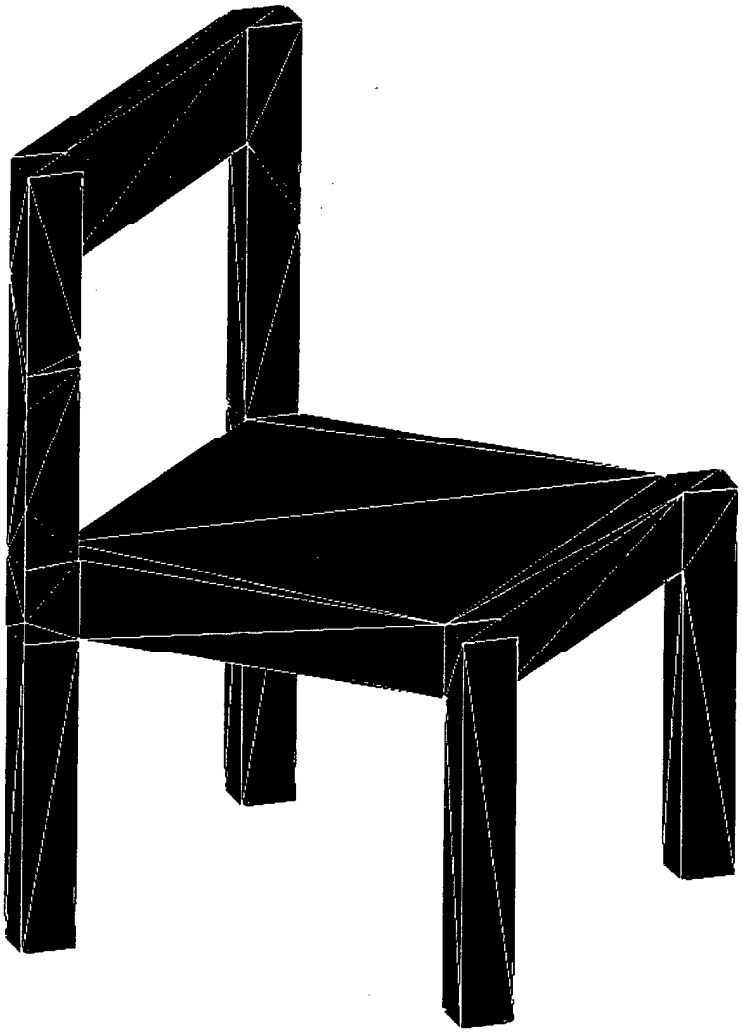


Figura 36. Corrección de la conformidad mediante inserción de puntos sobre las aristas.

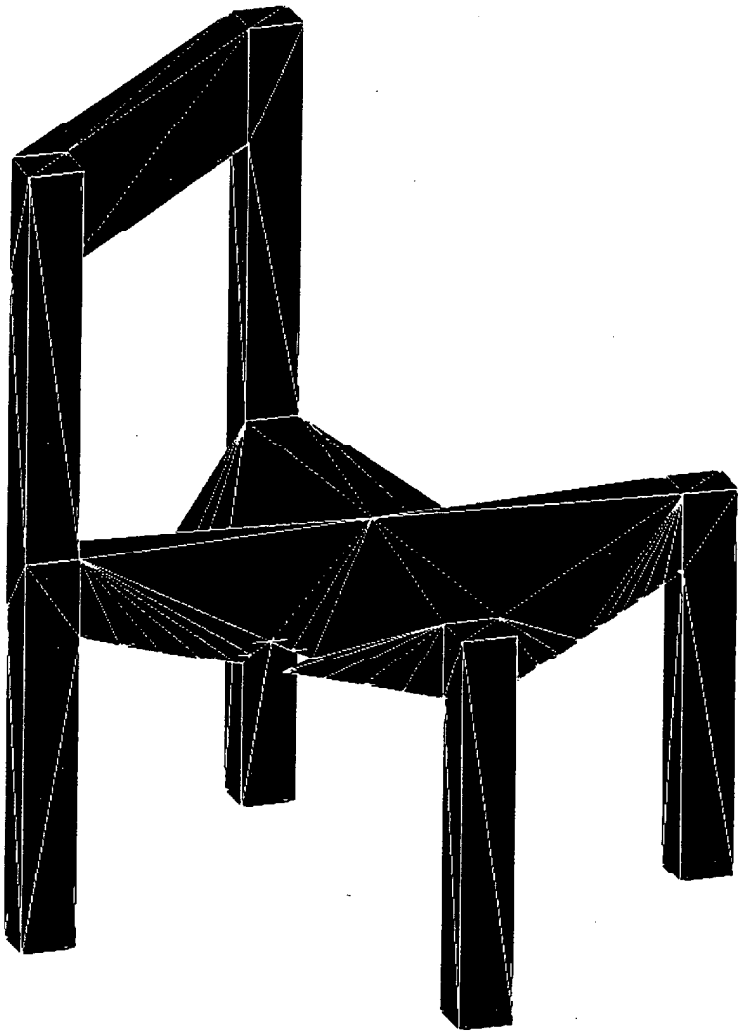


Figura 37. Otra situación de no conformidad.

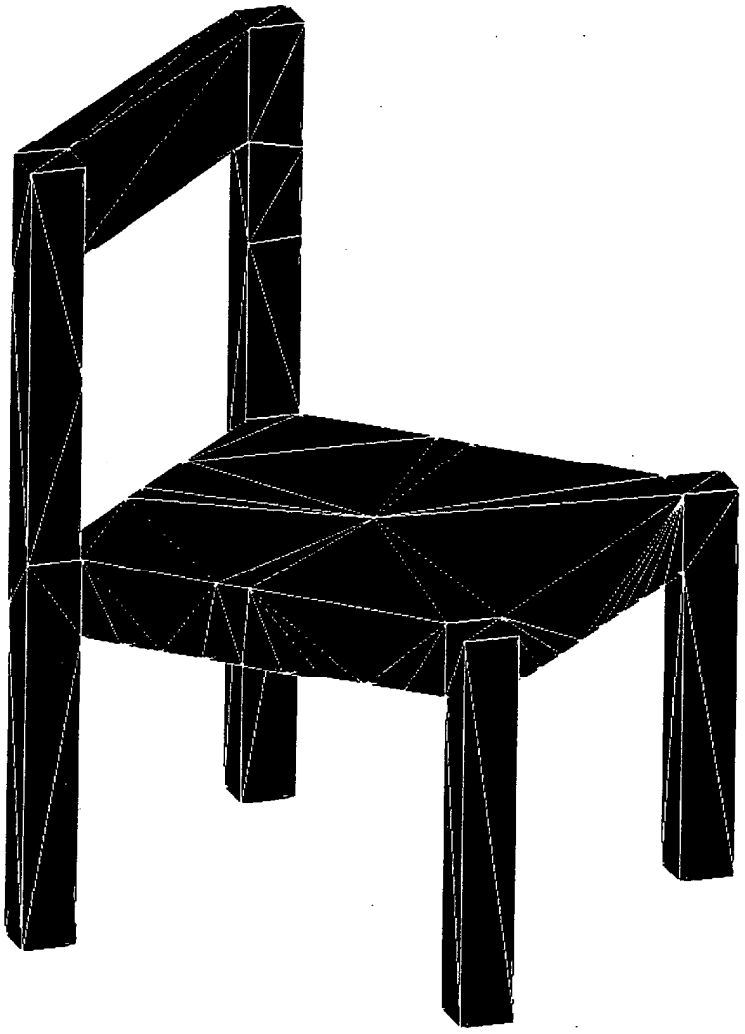


Figura 38. Corrección de la conformidad mediante inserción de puntos sobre las aristas.

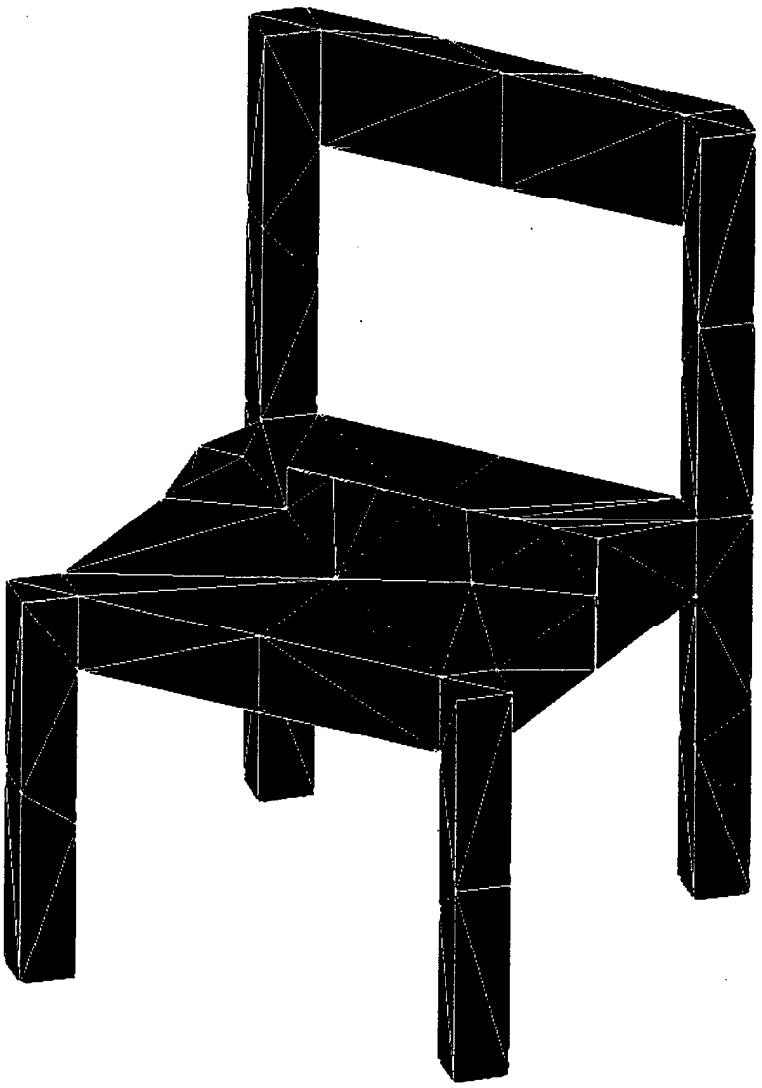


Figura 39. No conformidad entre la triangulación y el dominio causada por la colocación de una línea de puntos muy próxima al asiento de la silla.

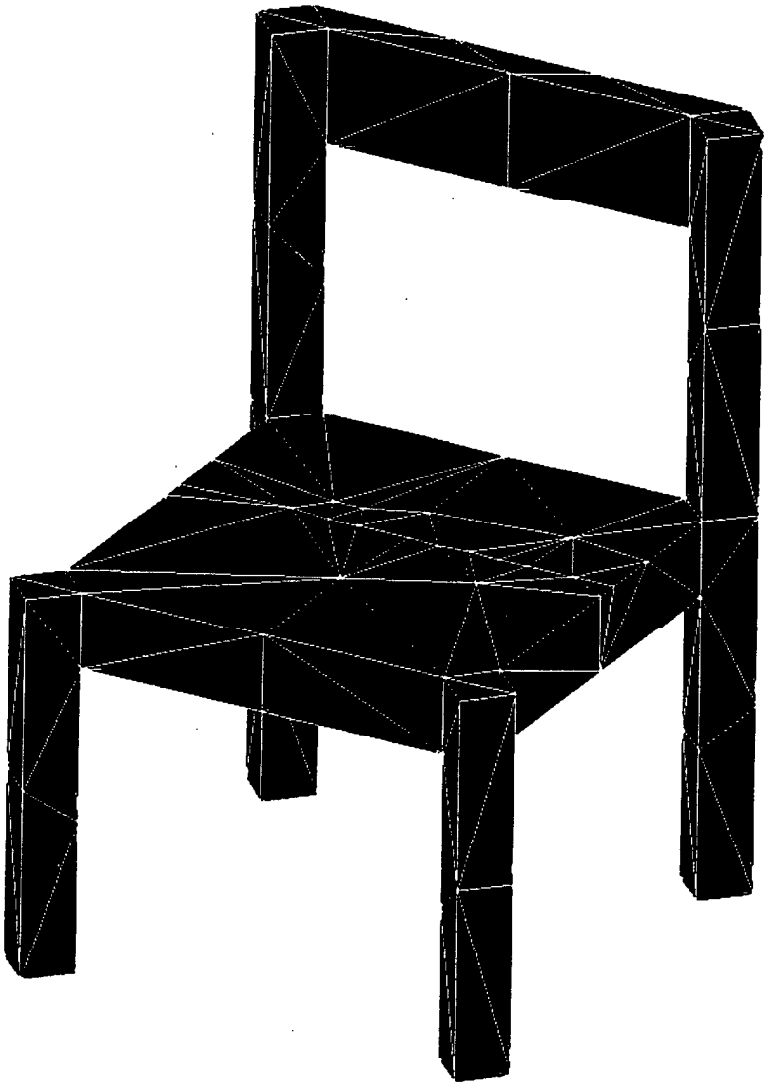


Figura 40. Corrección de la conformidad mediante inserción de puntos sobre las aristas y la cara.

3.7. Eliminación de los tetraedros exteriores al dominio.

Una vez conseguido que la triangulación sea conforme con la frontera del dominio debemos eliminar los tetraedros exteriores al mismo. Para ello se calcula el *centro de gravedad* de cada tetraedro y se analiza si éste es interior a alguno de los poliedros que componen el dominio. En caso afirmativo, dicho tetraedro es incluido en la lista de tetraedros del objeto. En este caso también hemos empleado el criterio del apartado 3.3.3. para establecer cuando el centro de un tetraedro está dentro de un determinado poliedro. En caso de estar interesados en cualquier otro elemento de la triangulación se podría proceder de manera similar a como se ha hecho con los tetraedros. El programa encargado de seleccionar los tetraedros del dominio dispone del fichero en el que se almacenan los datos de su geometría, el mismo del que dispone el programa de generación automática de puntos.

3.8. Medidas de calidad de la triangulación.

La triangulación de Delaunay tiene algunas propiedades, expuestas en el primer capítulo, que la hacen especialmente indicada para la aplicación del MEF. En todo caso, la calidad de la triangulación depende de la forma en que estén distribuidos los puntos. No existe una única medida de la calidad de una triangulación, entre otras cosas porque el concepto de calidad está estrechamente ligado al problema concreto que se pretenda resolver. En general, para el empleo del MEF interesa que los tetraedros sean lo más regulares posible. La regularidad de un tetraedro K se puede caracterizar de diversas maneras. Así, una posible medida de dicha regularidad está dada por el cociente entre el volumen de la esfera inscrita al tetraedro y el volumen del mismo.

$$C = \frac{\text{vol}(b_K)}{\text{vol}(K)}$$

Este cociente es máximo cuando el tetraedro es regular; su valor es $C_{eq} = \frac{\pi}{6\sqrt{3}}$. Existen otras medidas estrechamente relacionadas con la anterior; una es el cociente entre el volumen de la esfera inscrita y el volumen de la esfera circunscrita, siendo $C_{eq} = 1/3$ [BerEpp92].

$$C = \frac{\text{vol}(b_K)}{\text{vol}(B_K)}$$

Y otra, el cociente entre el radio de la esfera inscrita y el diámetro del mismo [Geor91].

$$C = \frac{\text{rad}(b_K)}{\text{diam}(K)}$$

También es posible dar una medida de la calidad de un tetraedro en función de los ángulos planos α_i, β_i y γ_i formados por las tres aristas concurrentes en cada vértice

[RivLev92]. El ángulo $\phi_K = \min_{1 \le i \le 4} \{\phi_i\}$ nos proporciona una buena medida de la calidad de un tetraedro, donde ϕ_i está dado por:

$$\phi_i = \text{sen}^{-1}(1 - \cos^2 \alpha_i - \cos^2 \beta_i - \cos^2 \gamma_i + 2\cos\alpha_i \cos\beta_i \cos\gamma_i)^{1/2}$$

Para los tetraedros planos ϕ_K vale 0 y para los tetraedros regulares su valor, $\pi/4$, es máximo.

A partir de las medidas de calidad de cada tetraedro se puede construir una medida de calidad de la triangulación. Por ejemplo, el parámetro F definido por:

$$F = \frac{1}{n} \sum_{i=1}^{n} (1 - \frac{C_i}{C_{eq}})$$

donde n es el número total de tetraedros y C_{eq} es el valor de C para los tetraedros equiláteros, nos da una idea global de la calidad de la triangulación. El valor óptimo de F es cero y ocurre cuando todos los tetraedros son regulares [AllFig92].

No obstante, la calidad de una triangulación no queda bien reflejada por una medida de tipo promedio como F; resulta más adecuado hacer una clasificación por grupos que tengan medidas de regularidad parecidas. Supongamos que la medida de la regularidad de un tetraedro está comprendida entre los valores C_0 y C_k, y que $C_0 < C_1 < \ldots < C_{k-1} < C_k$ es una partición del intervalo $[C_0, C_k]$. Mediante un diagrama de barras o una estructura similar podemos indicar el número de tetraedros cuya medida de regularidad está comprendida entre los valores C_{i-1} y C_i $(1 \le i \le k)$. Este diagrama nos proporciona una información bastante completa de la calidad de la triangulación. En nuestro caso hemos representado el ángulo ϕ_K en un diagrama de barras que divide el intervalo $[0, \pi/4]$ en siete partes. En el capítulo cuarto se mostrarán los diagramas correspondientes a varias aplicaciones.

4. RESULTADOS Y APLICACIONES

4.1. Aplicaciones y características del programa de triangulación.

Resulta complicado hacer una estimación de la complejidad algorítmica del programa de triangulación. Factores como el número de tetraedros que se eliminan cada vez que se añade un nuevo punto a la triangulación, el número de tetraedros recorridos hasta encontrar el núcleo, o el número de iteraciones que necesita el procedimiento CONJUNTO D_1^i para converger, influyen substancialmente en la complejidad del programa y son difícilmente predecibles. En su lugar hemos preferido aportar algunos datos estadísticos obtenidos en diversas aplicaciones prácticas. Los datos de cada aplicación se aportan en dos tablas y seis gráficas. La primera tabla contiene datos generales de la triangulación. La segunda explica el comportamiento del procedimiento CONJUNTO D_1^i. Las cuatro primeras gráficas se centran en las características del procedimiento de búsqueda del núcleo. La penúltima gráfica muestra el tiempo de CPU requerido por el programa para triangular el conjunto de puntos y la última analiza la calidad de la triangulación.

4.1.1. Aplicación: letra.

APLICACIÓN: LETRA
N° de puntos en el objeto: 10.440
N° total de puntos en la triangulación: 10.510
N° de tetraedros del objeto: 51.975
N° total de tetraedros en la triangulación: 60.754

Tabla 1.

Características del procedimiento CONJUNTO D_1^i		
Valor inicial de ε: 0'002	Valor inicial de ε: 0'001	Valor inicial de ε: 0'0001
N° máximo de iteraciones necesaria para converger sin cambiar ε: 11	N° máximo de iteraciones necesaria para converger sin cambiar ε: 11	N° máximo de iteraciones necesaria para converger sin cambiar ε: 11
N° promedio de iteraciones necesaria para converger sin cambiar ε: 1'9352	N° promedio de iteraciones necesaria para converger sin cambiar ε: 1'92883	N° promedio de iteraciones necesaria para converger sin cambiar ε: 1'92702
N° máximo de cambios $\varepsilon \rightarrow \varepsilon/5$ necesarios para converger: 0	N° máximo de cambios $\varepsilon \rightarrow \varepsilon/5$ necesarios para converger: 0	N° máximo de cambios $\varepsilon \rightarrow \varepsilon/5$ necesarios para converger: 0
N° máximo de tetraedros en D_1^i: 191	N° máximo de tetraedros en D_1^i: 191	N° máximo de tetraedros en D_1^i: 191
N° promedio de tetraedros en D_1^i: 32'9942	N° promedio de tetraedros en D_1^i: 33'0078	N° promedio de tetraedros en D_1^i: 33'01493

Tabla 2.

El valor inicial de ε, que salvo indicación en sentido contrario hemos tomado igual a 0'002, es el valor mínimo que inicialmente admitimos para construir un conjunto D_1^i ε-en forma de estrella. En algunos casos este valor inicial es demasiado restrictivo, por lo que debe ser cambiado por uno menor. El cambio consiste en dividir el valor actual de ε por cinco (sin sobrepasar en ningún caso el valor ε_N) hasta que el proceso concluya. Como se explicó en el capítulo segundo, la construcción de D_1^i se efectúa mediante una serie de *expansiones* sucesivas a partir del núcleo. Cuando en la tabla anterior hablamos de número máximo o promedio de iteraciones que necesita el procedimiento CONJUNTO D_1^i para converger, nos referimos al correspondiente número de *expansiones* necesarias. Si todos los tetraedros cuyas esferas contienen al punto x_{i+1} forman un conjunto ε-en forma de estrella, para el valor inicial de ε, el procedimiento sólo necesita una iteración para converger; en caso contrario necesitará más de una.

En las cuatro gráficas siguientes se exponen algunos resultados referentes al proceso de búsqueda de los tetraedros que forman el núcleo. En las dos primeras se

muestra el número máximo de tetraedros que se han recorrido hasta encontrar el núcleo en relación al número de tetraedros existentes en la triangulación en ese momento. Los datos han sido tomados cada vez que se producía un aumento en el número máximo de tetraedros recorridos. La primera gráfica corresponde al proceso de búsqueda cuando éste comienza por el último tetraedro generado hasta ese momento. En la segunda, el proceso siempre empieza en el mismo tetraedro; concretamente, en el número uno.

Curiosamente, en todas las aplicaciones observamos que cuando se comienza la búsqueda del núcleo por el último tetraedro es necesario recorrer más tetraedros, en el caso más desfavorable, que cuando se comienza por uno fijo, en este caso por el número uno. Sin embargo, los valores promedio son siempre más favorables cuando se empieza por el último tetraedro. En las dos gráficas siguientes se muestran los valores medios, en función del número de puntos añadidos a la triangulación, del número de tetraedros recorridos hasta alcanzar el núcleo, empezando por el último y por el número uno, respectivamente.

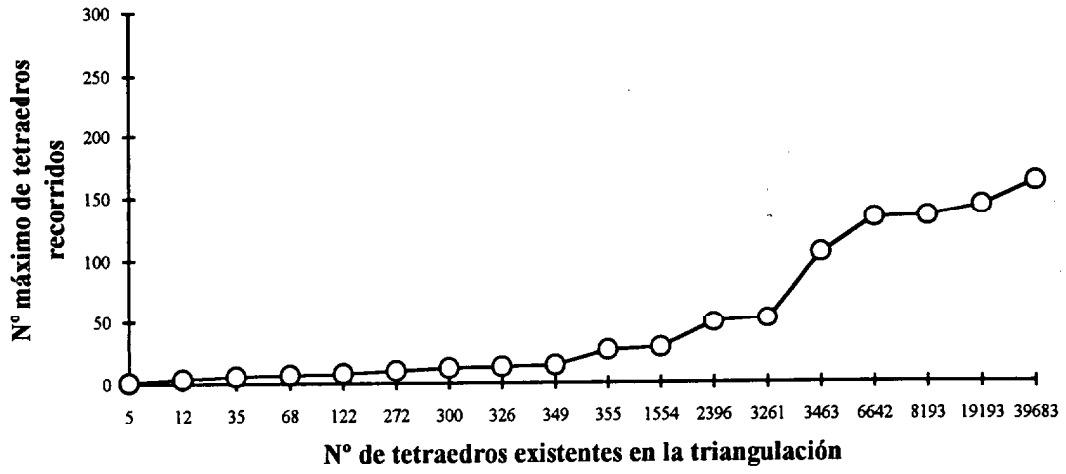


Figura 41. Máximo nº de tetraedros recorridos, comenzando por el último de los generados, para encontrar el núcleo.

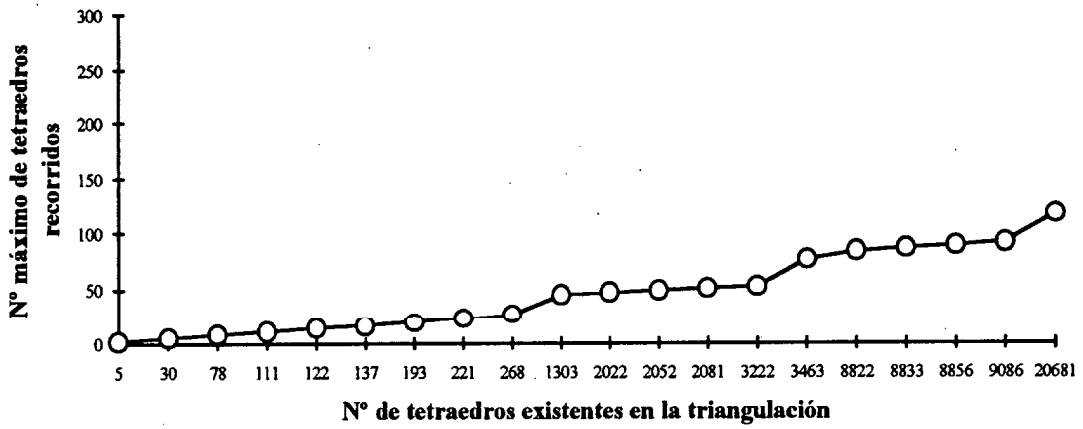


Figura 42. Máximo nº de tetraedros recorridos, comenzando por el nº 1, para encontrar el núcleo.

Notese que a partir del valor máximo de tetraedros que figura en el eje de abscisas de las gráficas anteriores no se ha producido ningún aumento en el número de tetraedros recorridos.

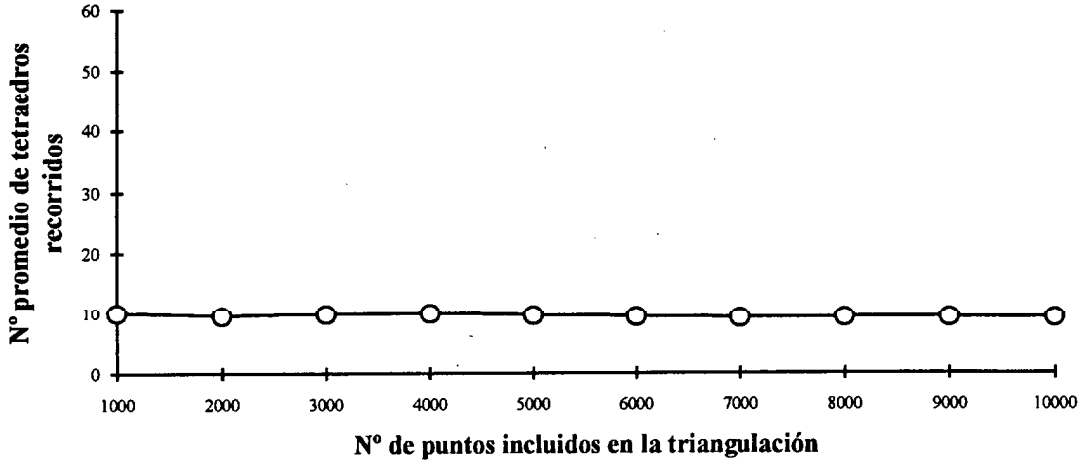


Figura 43. Nº promedio de tetraedros recorridos, comenzando por el último, para encontrar el núcleo.

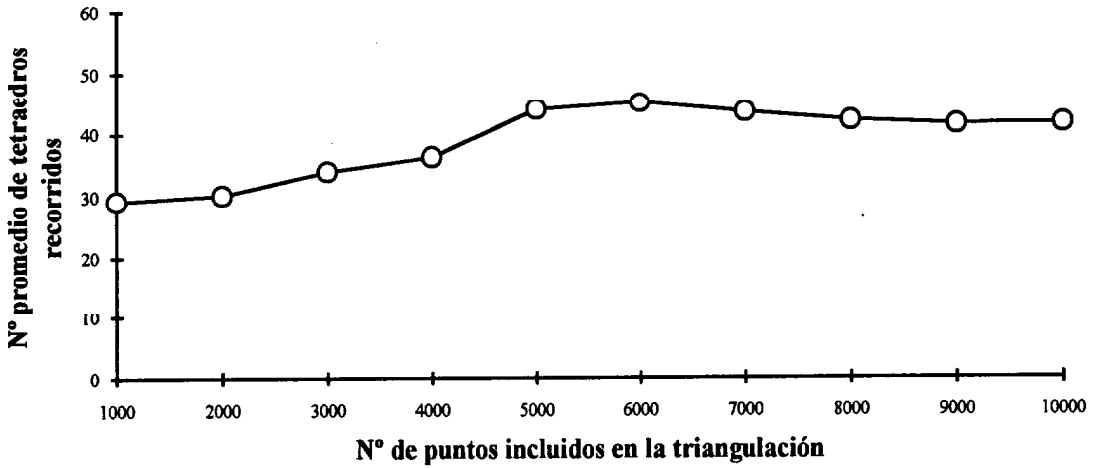


Figura 44. Nº promedio de tetraedros recorridos, comenzando por el nº 1, para encontrar el núcleo.

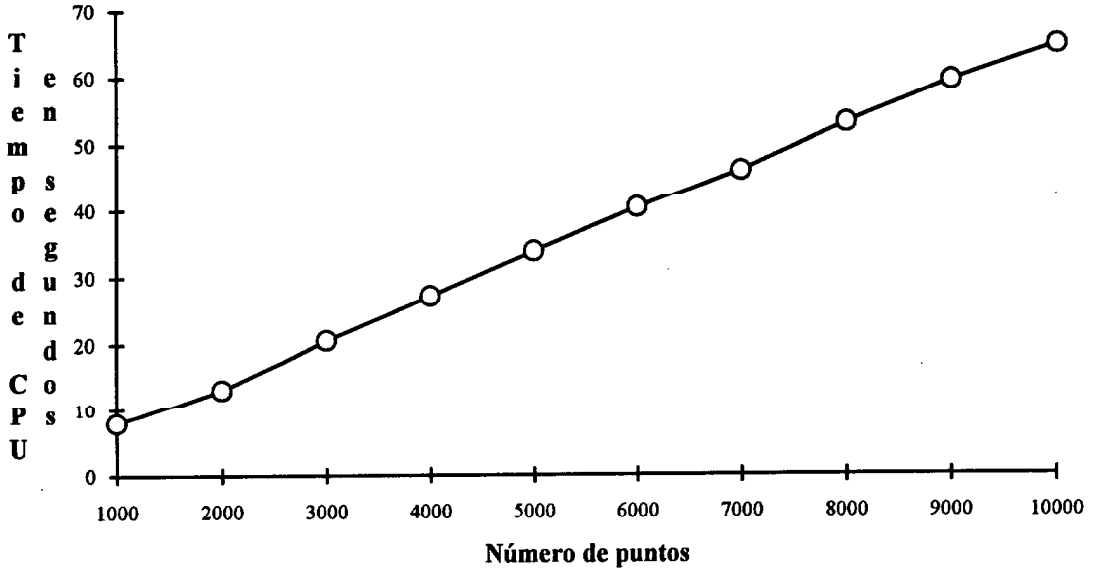


Figura 45. Tiempo de CPU en función del número de puntos incluidos.

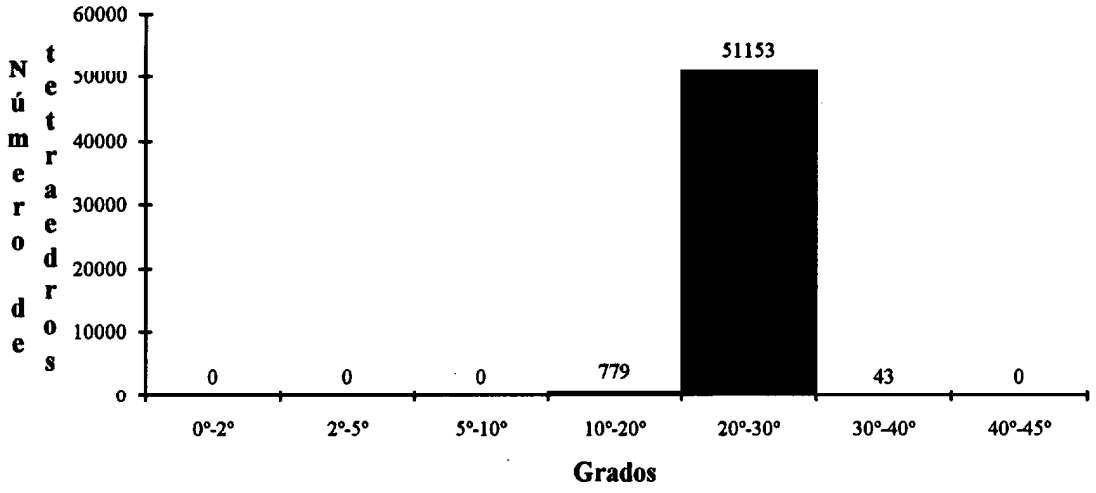


Figura 46. Medida de la calidad de la triangulación del objeto.

En las aplicaciones siguientes aportaremos el mismo tipo de datos que en la aplicación anterior.

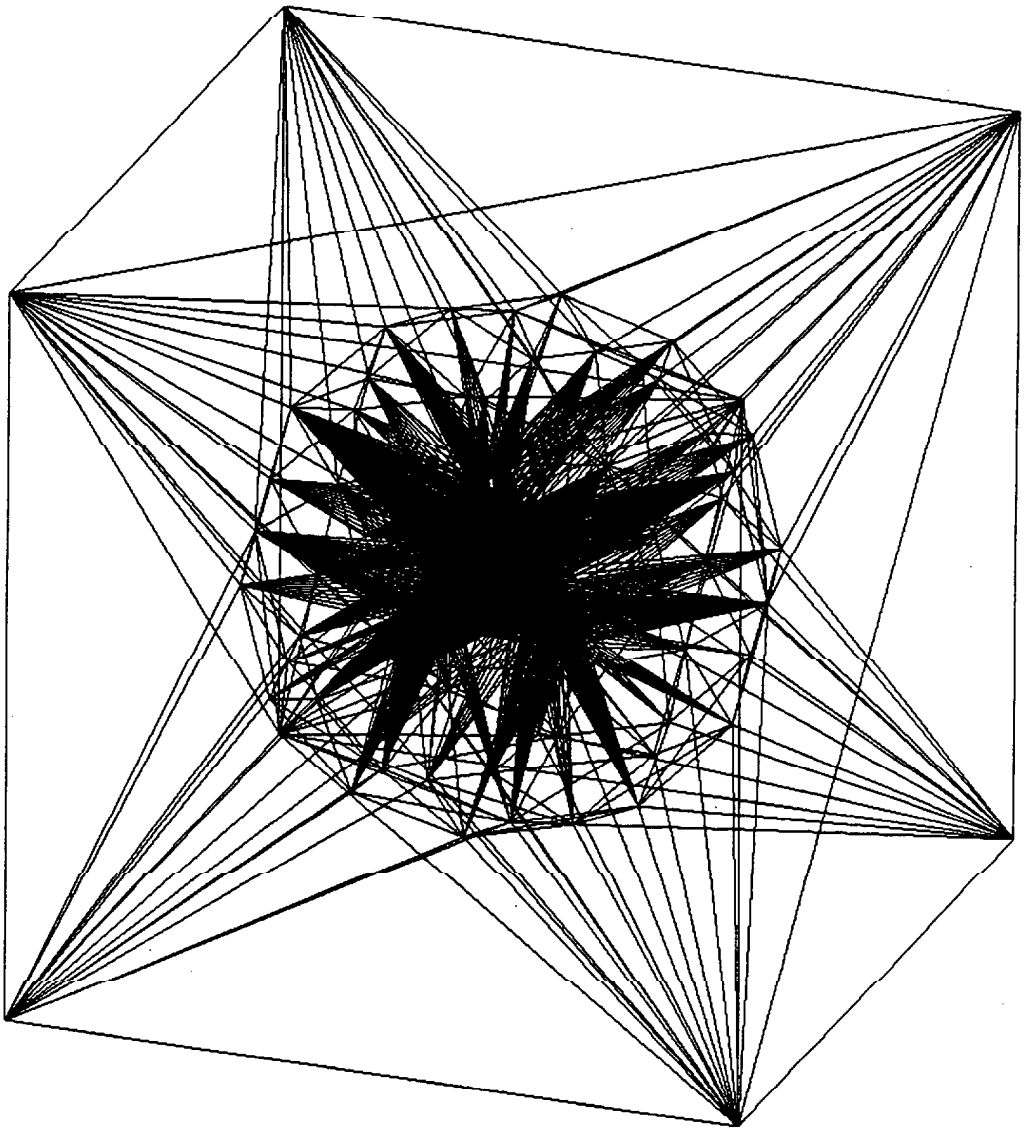


Figura 47. Triangulación global: puntos del dominio más puntos de la superficie esférica y del cubo envolvente.

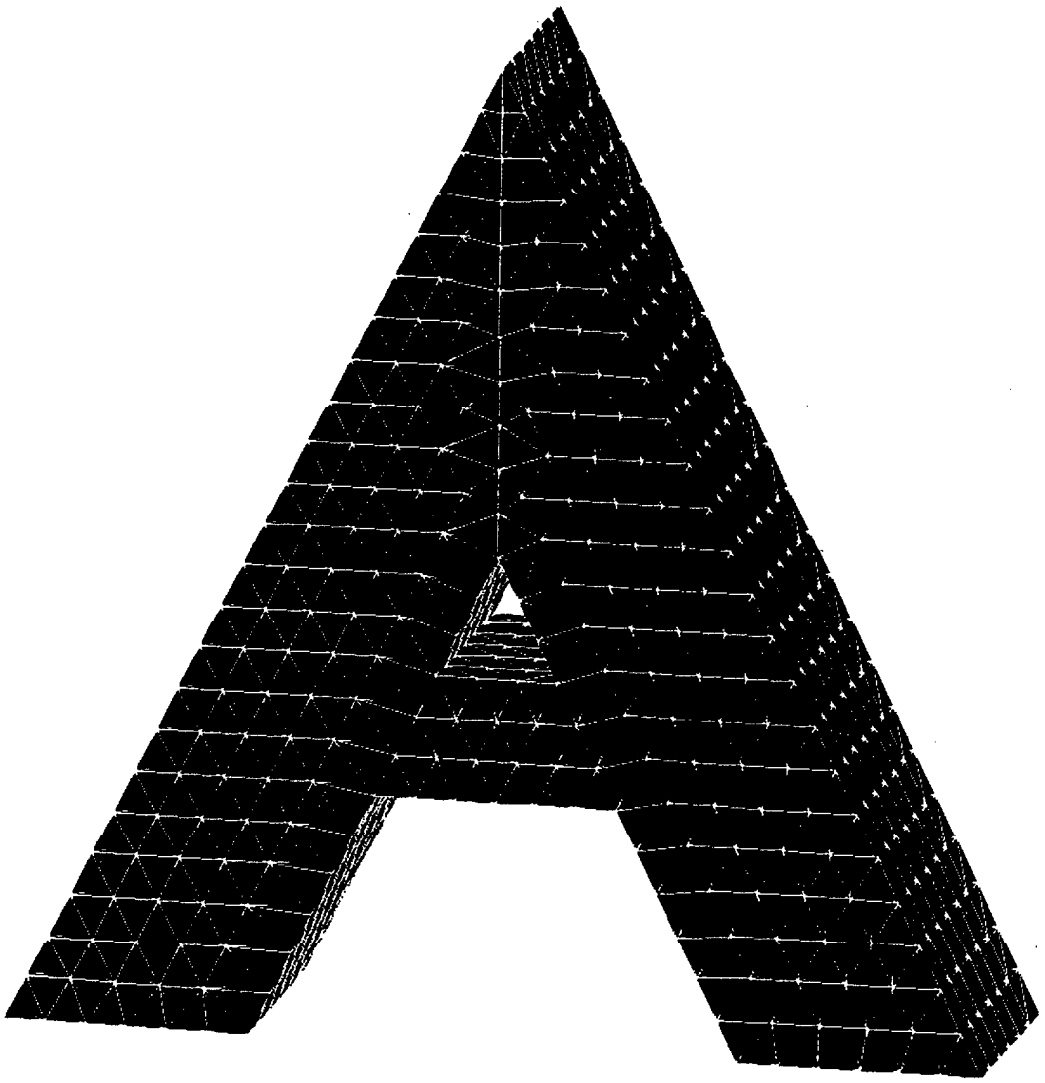


Figura 48. Triangulación del dominio con una densidad uniforme de puntos.

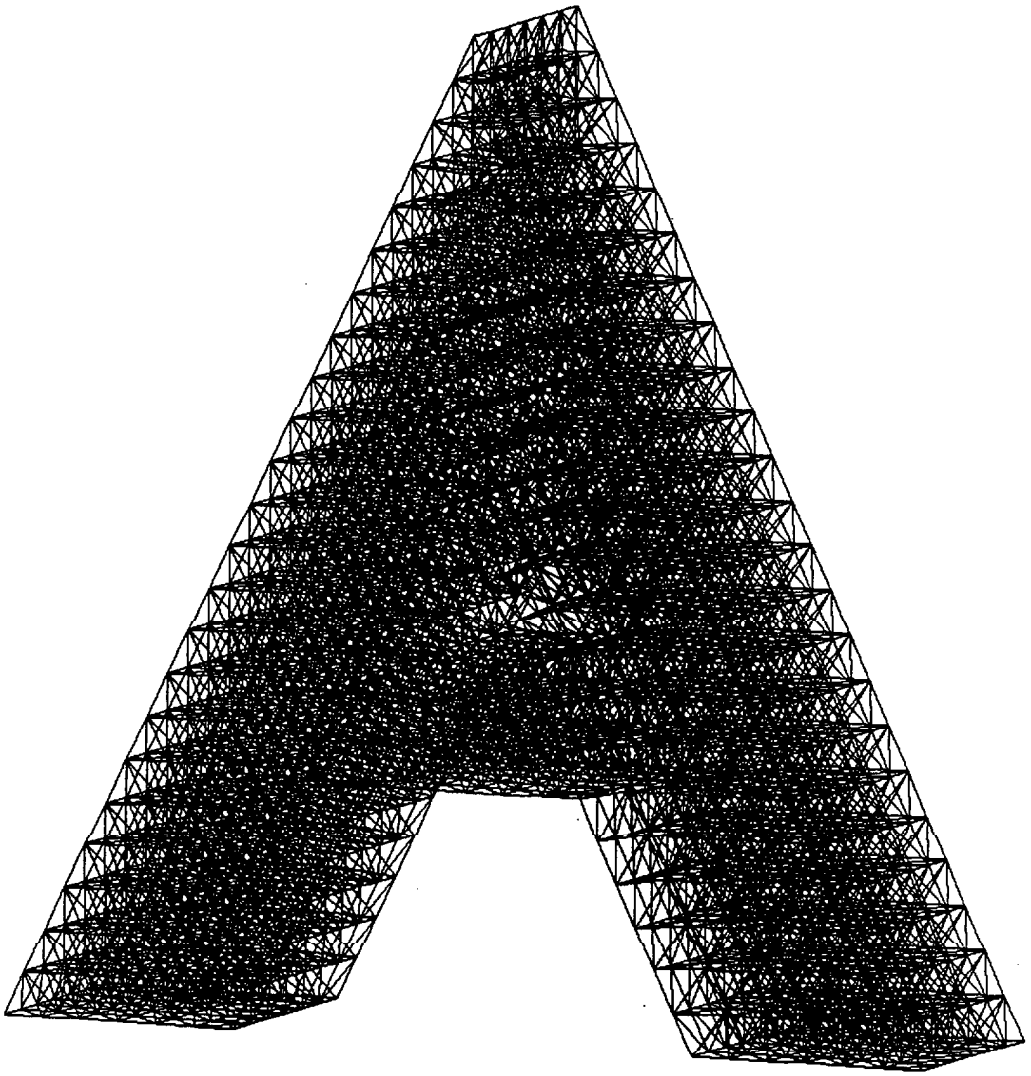


Figura 49. Lineas de la triangulación anterior.

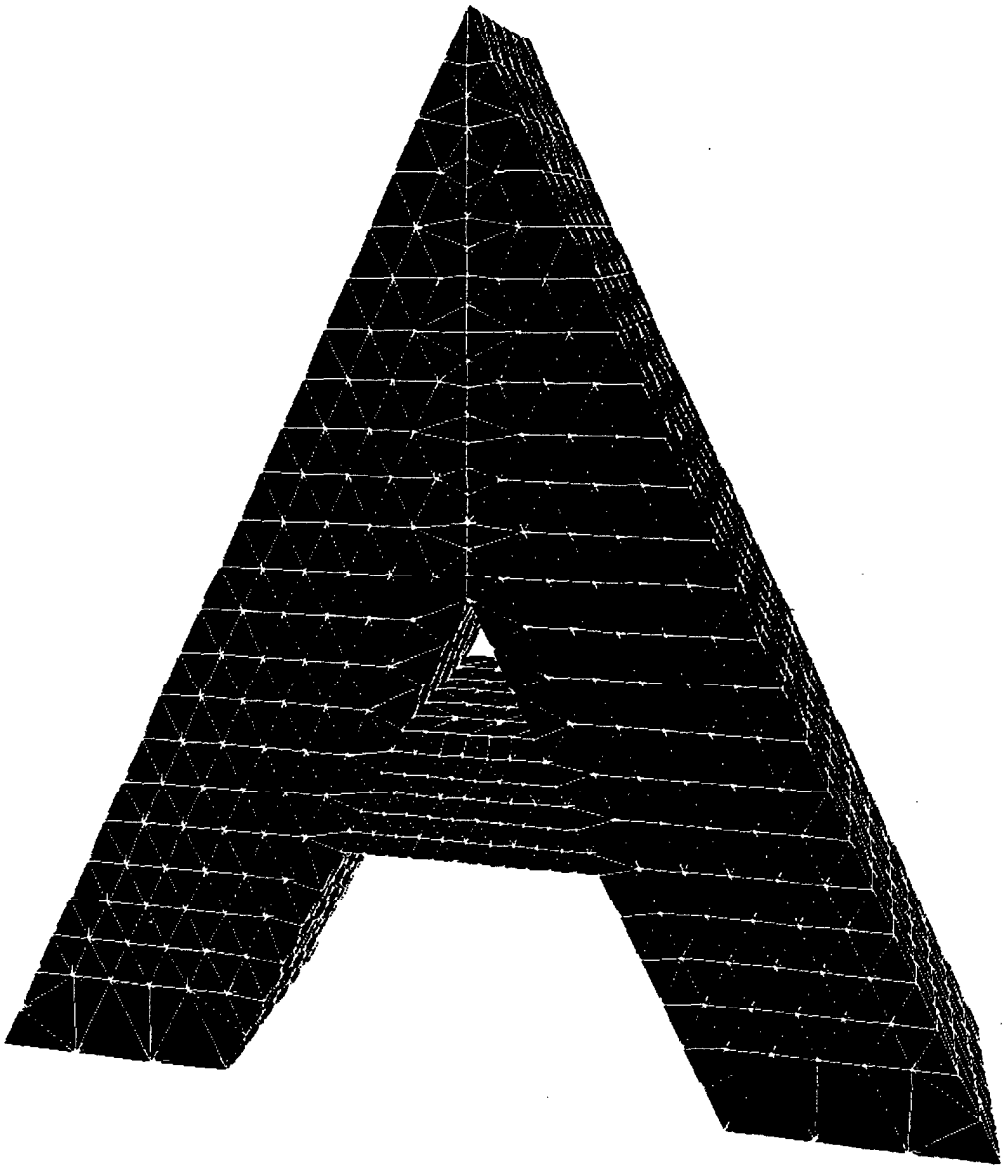


Figura 50. Triangulación con densidad de puntos no uniforme.

4.1.2. Aplicación: silla.

APLICACIÓN: SILLA
N° de puntos en el objeto: 7.046
N° total de puntos en la triangulación: 7.116
N° de tetraedros del objeto: 30.143
N° total de tetraedros en la triangulación: 43.478

Tabla 3.

Características del procedimiento CONJUNTO D_1^i		
Valor inicial de ε: 0'002	Valor inicial de ε: 0'001	Valor inicial de ε: 0'0001
N° máximo de iteraciones necesaria para converger sin cambiar ε: 15	N° máximo de iteraciones necesaria para converger sin cambiar ε: 16	N° máximo de iteraciones necesaria para converger sin cambiar ε: 12
N° promedio de iteraciones necesaria para converger sin cambiar ε: 1'58614	N° promedio de iteraciones necesaria para converger sin cambiar ε: 1'56577	N° promedio de iteraciones necesaria para converger sin cambiar ε: 1'50155
N° máximo de cambios $\varepsilon \to \varepsilon/5$ necesarios para converger: 1	N° máximo de cambios $\varepsilon \to \varepsilon/5$ necesarios para converger: 0	N° máximo de cambios $\varepsilon \to \varepsilon/5$ necesarios para converger: 0
N° máximo de tetraedros en D_1^i: 387	N° máximo de tetraedros en D_1^i: 389	N° máximo de tetraedros en D_1^i: 389
N° promedio de tetraedros en D_1^i: 41'65092	N° promedio de tetraedros en D_1^i: 41'74367	N° promedio de tetraedros en D_1^i: 41'92594

Tabla 4.

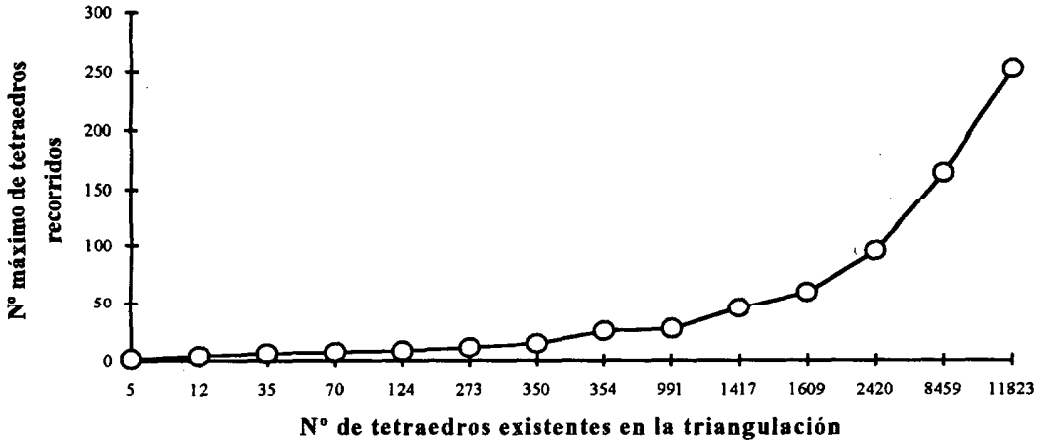


Figura 51. Máximo nº de tetraedros recorridos, comenzando por el último de los generados, para encontrar el núcleo.

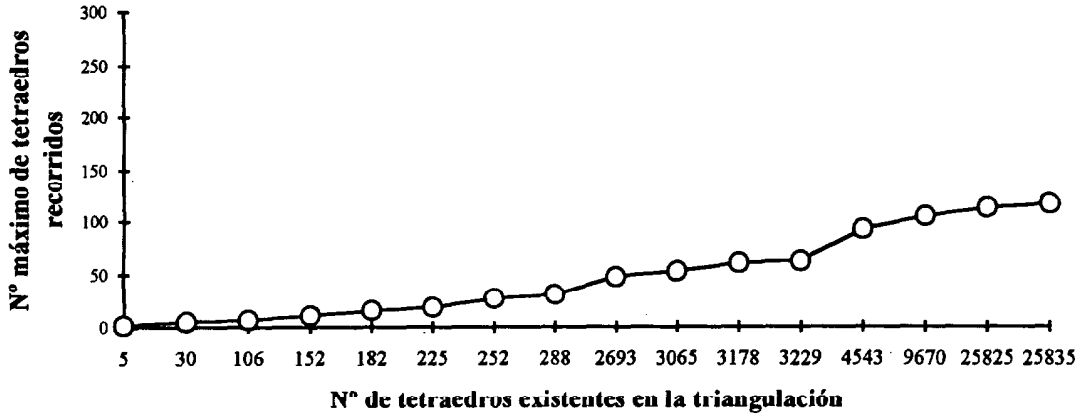


Figura 52. Máximo nº de tetraedros recorridos, comenzando por el nº 1, para encontrar el núcleo.

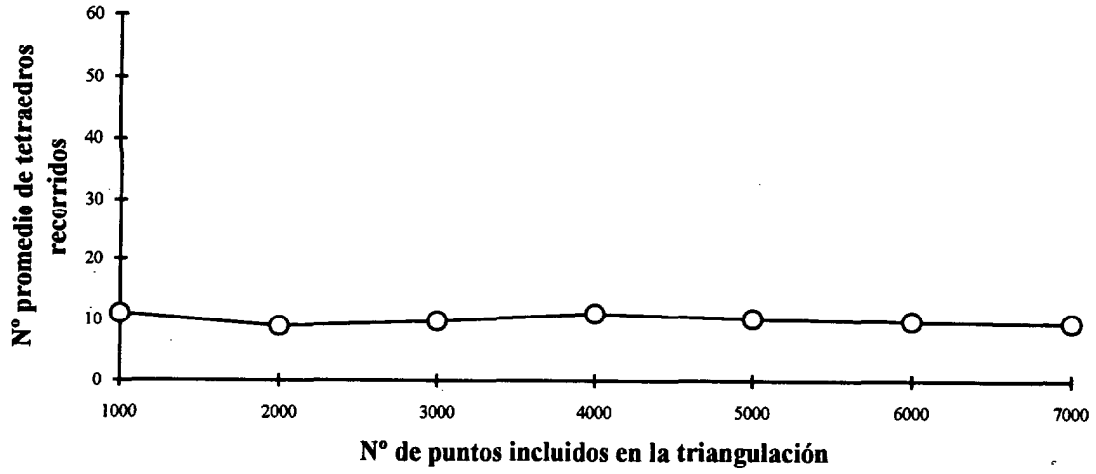


Figura 53. Nº promedio de tetraedros recorridos, comenzando por el último, para encontrar el núcleo.

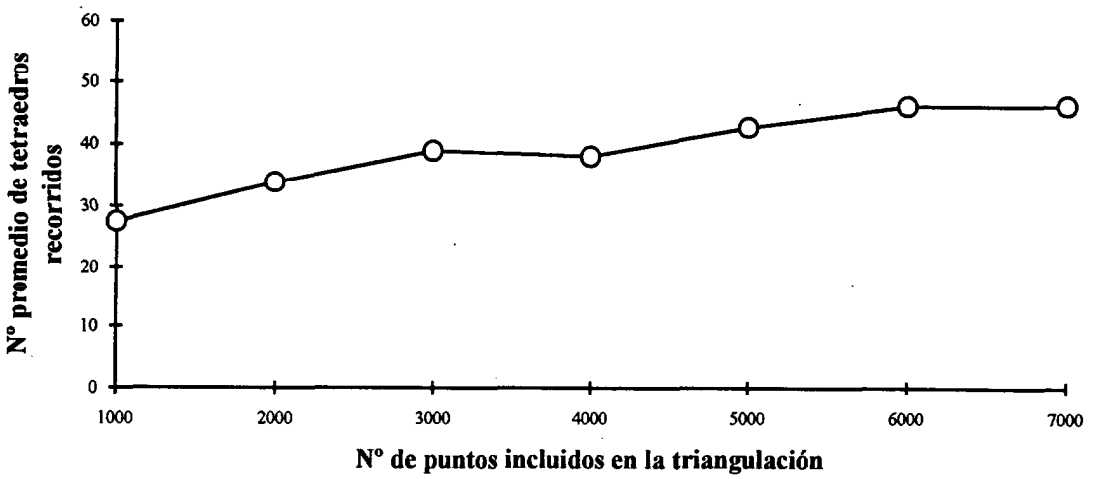


Figura 54. Nº promedio de tetraedros recorridos, comenzando por el nº 1, para encontrar el núcleo.

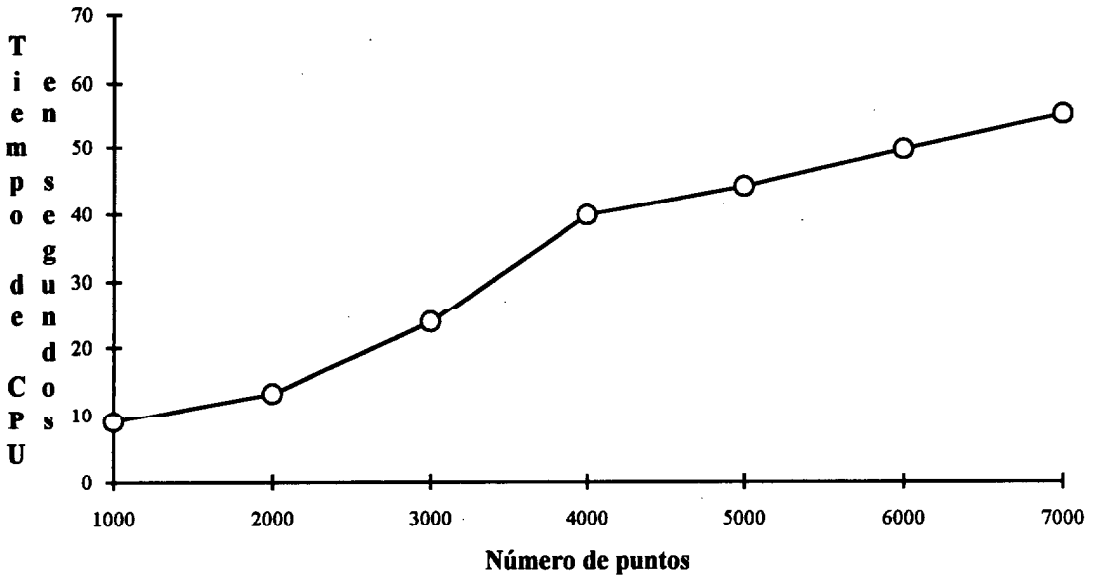


Figura 55. Tiempo de CPU en función del número de puntos incluidos.

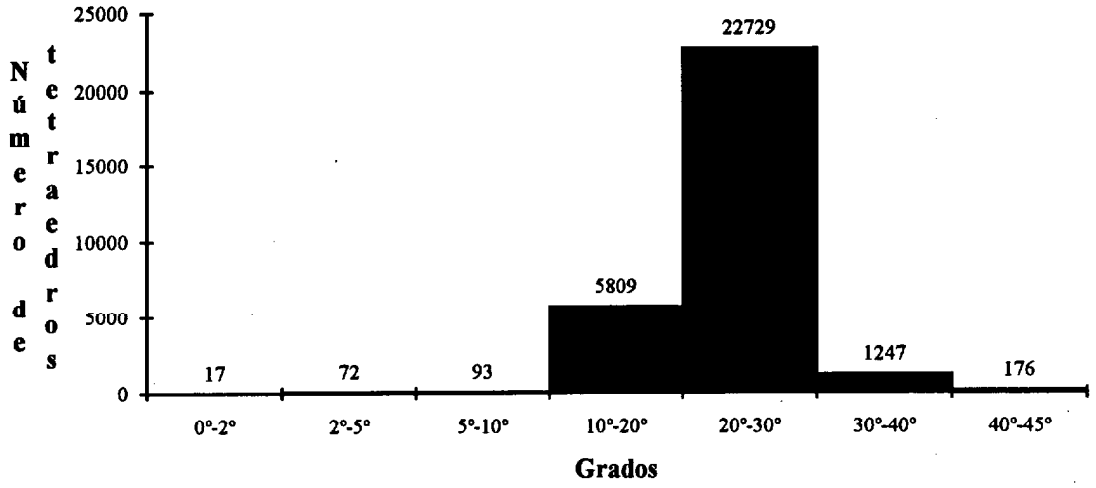


Figura 56. Medida de la calidad de la triangulación del objeto.

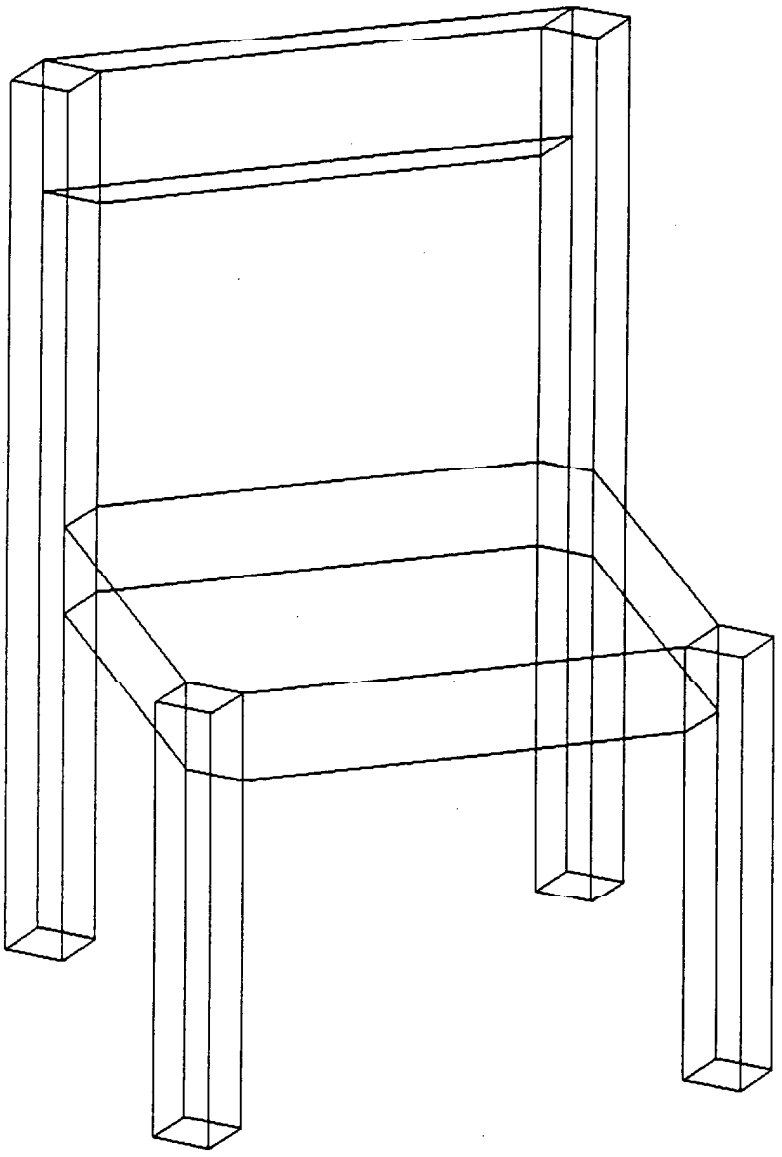


Figura 57. Vértices, aristas, caras y regiones que definen el objeto.

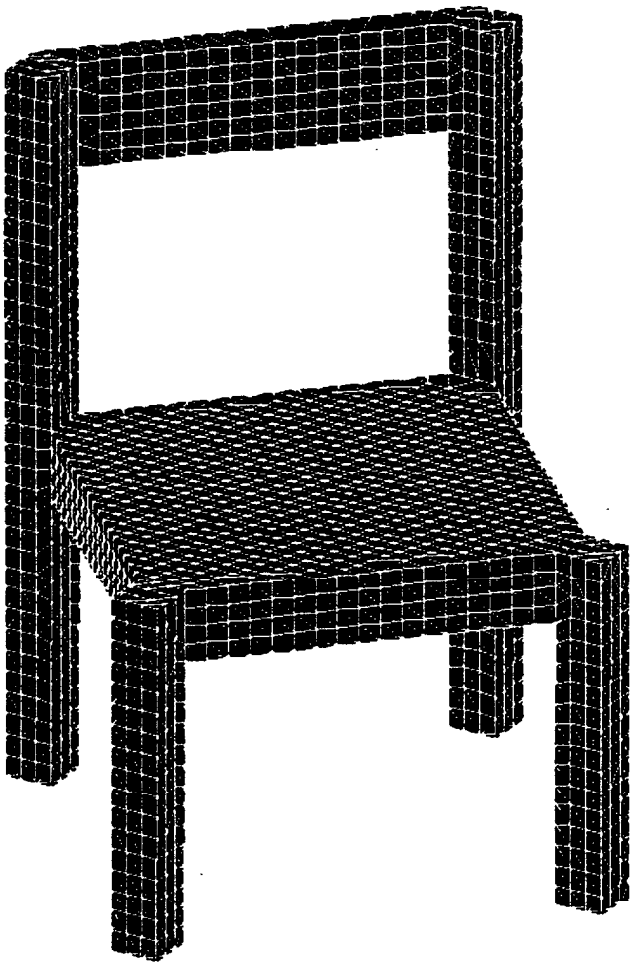


Figura 58. Triangulación del dominio con densidad uniforme de puntos.

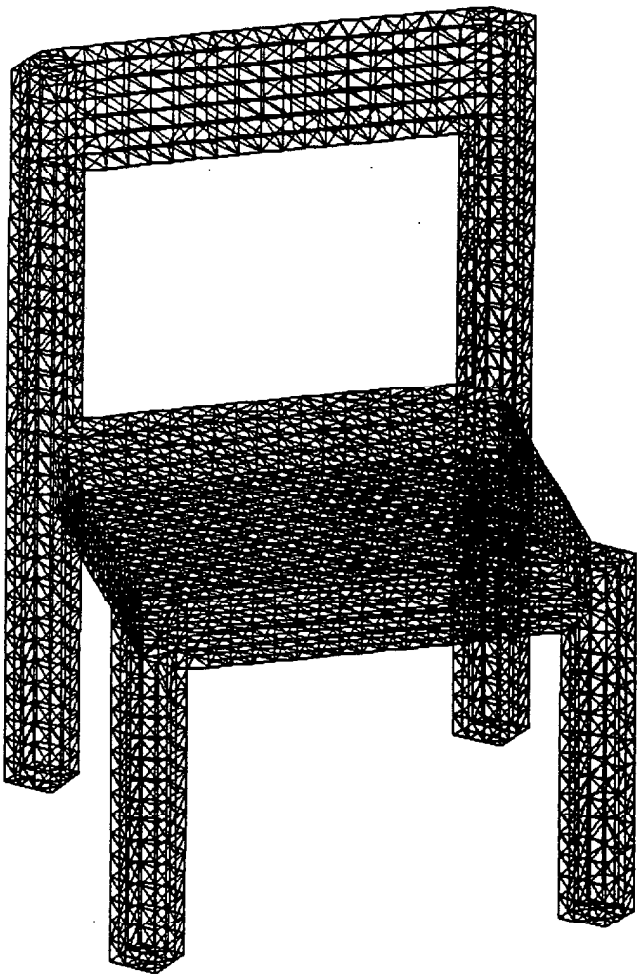


Figura 59. Líneas de la triangulación de la superficie.

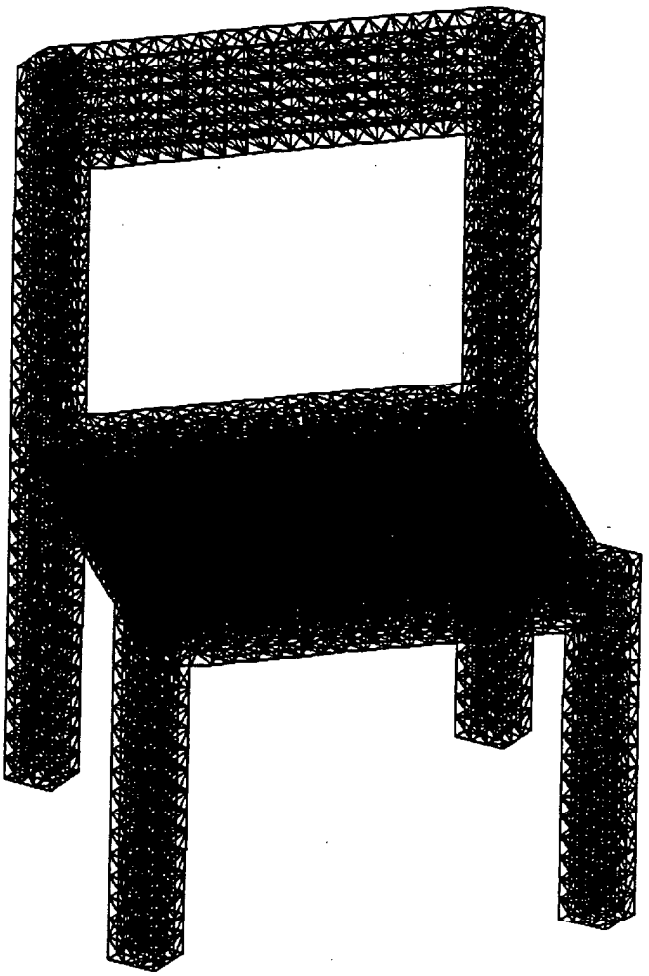


Figura 60. Líneas de la triangulación de la superficie y el interior.

4.1.3. Aplicación: puente.

APLICACIÓN: PUENTE
N° de puntos en el objeto: 10.051
N° total de puntos en la triangulación: 10.121
N° de tetraedros del objeto: 38.004
N° total de tetraedros en la triangulación: 63.693

Tabla 5.

Características del procedimiento CONJUNTO D_1^i		
Valor inicial de ε: 0'002	Valor inicial de ε: 0'001	Valor inicial de ε: 0'0001
N° máximo de iteraciones necesaria para converger sin cambiar ε: 24	N° máximo de iteraciones necesaria para converger sin cambiar ε: 24	N° máximo de iteraciones necesaria para converger sin cambiar ε: 24
N° promedio de iteraciones necesaria para converger sin cambiar ε: 1'66703	N° promedio de iteraciones necesaria para converger sin cambiar ε: 1'55726	N° promedio de iteraciones necesaria para converger sin cambiar ε: 1'53275
N° máximo de cambios $\varepsilon \rightarrow \varepsilon/5$ necesarios para converger: 1	N° máximo de cambios $\varepsilon \rightarrow \varepsilon/5$ necesarios para converger: 1	N° máximo de cambios $\varepsilon \rightarrow \varepsilon/5$ necesarios para converger: 0
N° máximo de tetraedros en D_1^i: 348	N° máximo de tetraedros en D_1^i: 354	N° máximo de tetraedros en D_1^i: 354
N° promedio de tetraedros en D_1^i: 29'6380	N° promedio de tetraedros en D_1^i: 29'7843	N° promedio de tetraedros en D_1^i: 29'8469

Tabla 6.

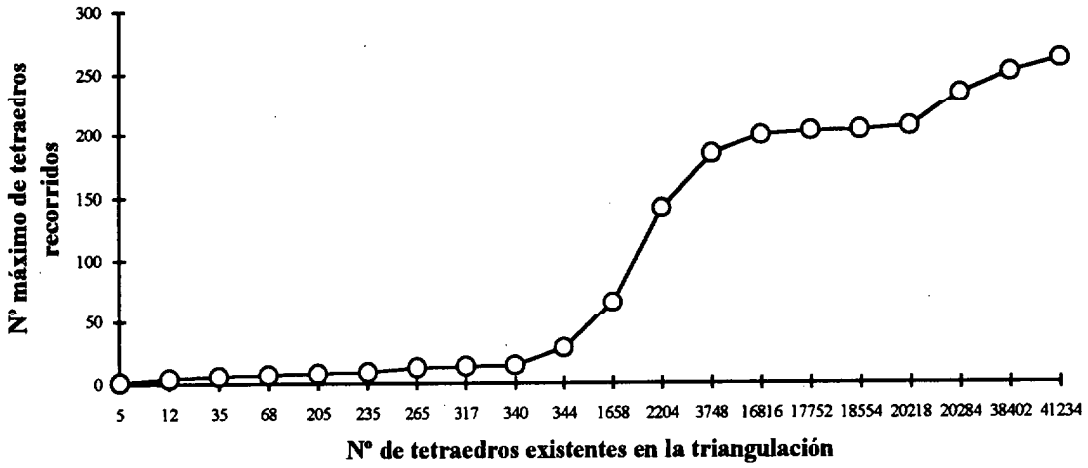


Figura 61. Máximo nº de tetraedros recorridos, comenzando por el último de los generados, para encontrar el núcleo.

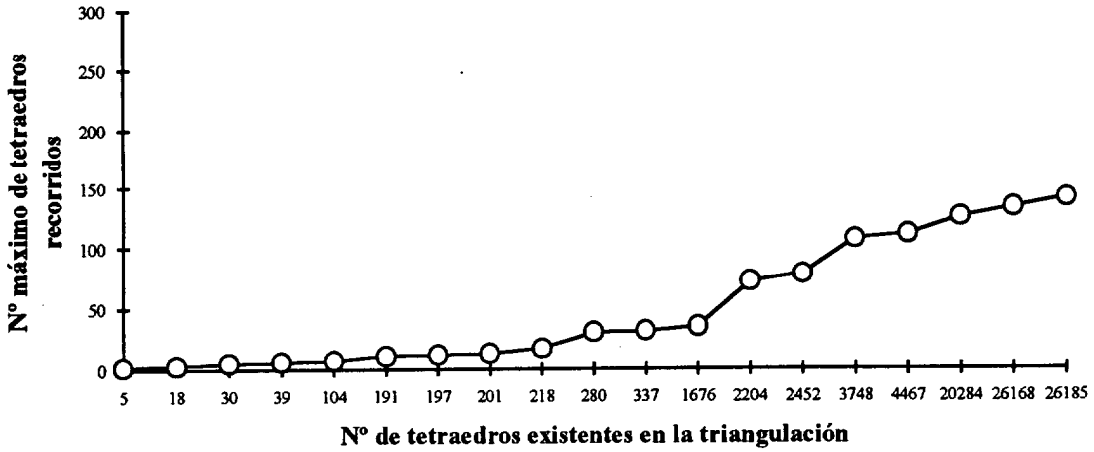


Figura 62. Máximo nº de tetraedros recorridos, comenzando por el nº 1, para encontrar el núcleo.

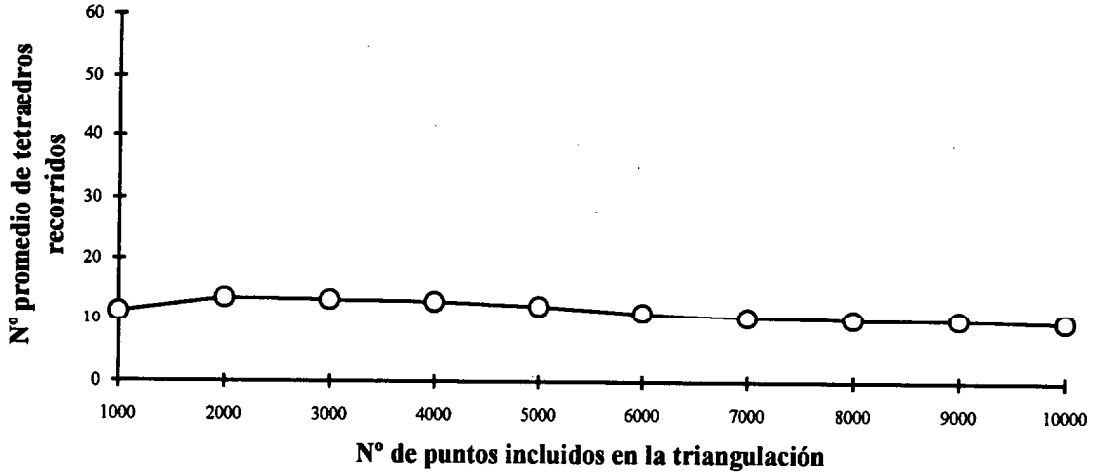


Figura 63. Nº promedio de tetraedros recorridos, comenzando por el último, para encontrar el núcleo.

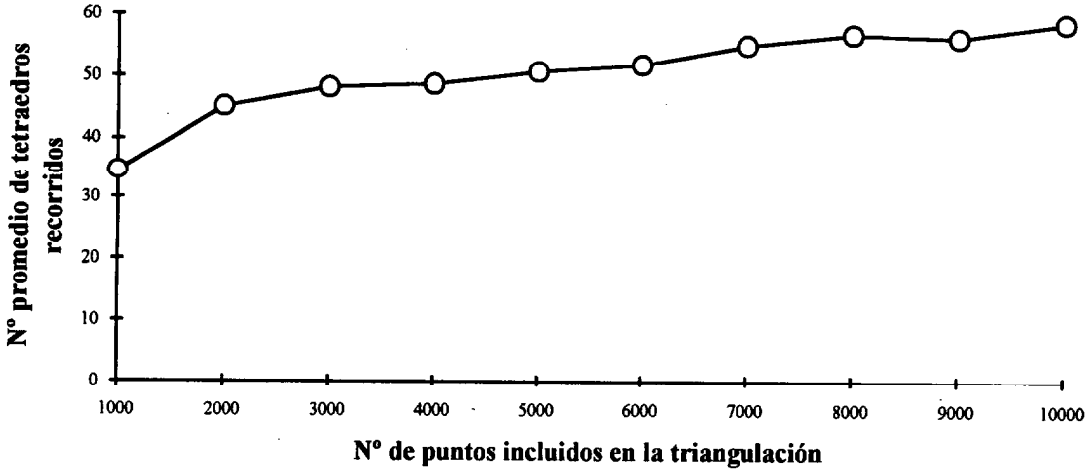


Figura 64. Nº promedio de tetraedros recorridos, comenzando por el nº 1, para encontrar el núcleo.

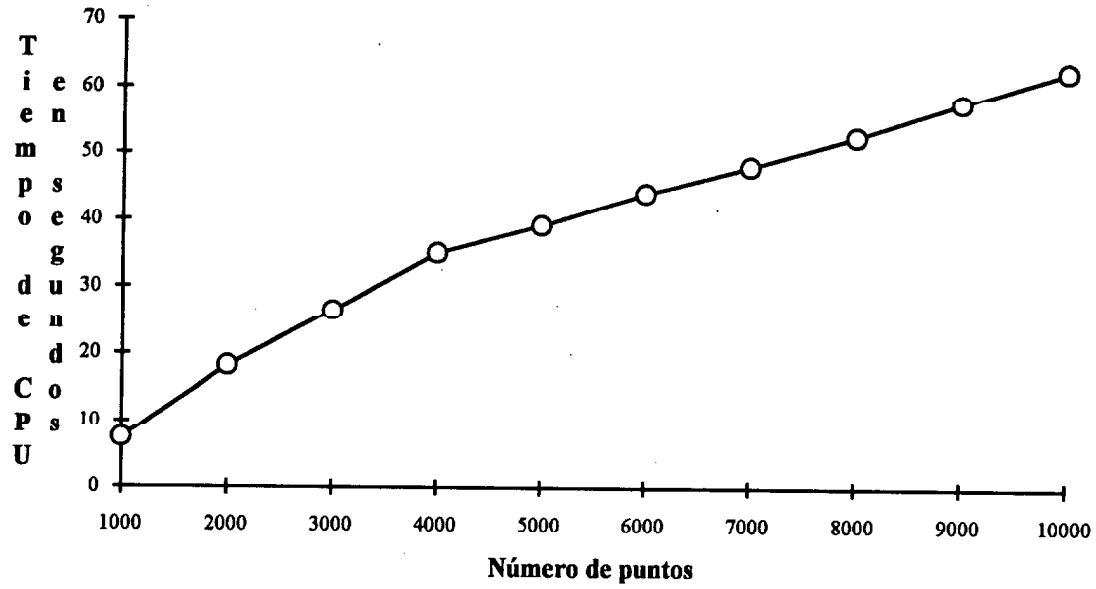


Figura 65. Tiempo de CPU en función del número de puntos incluidos.

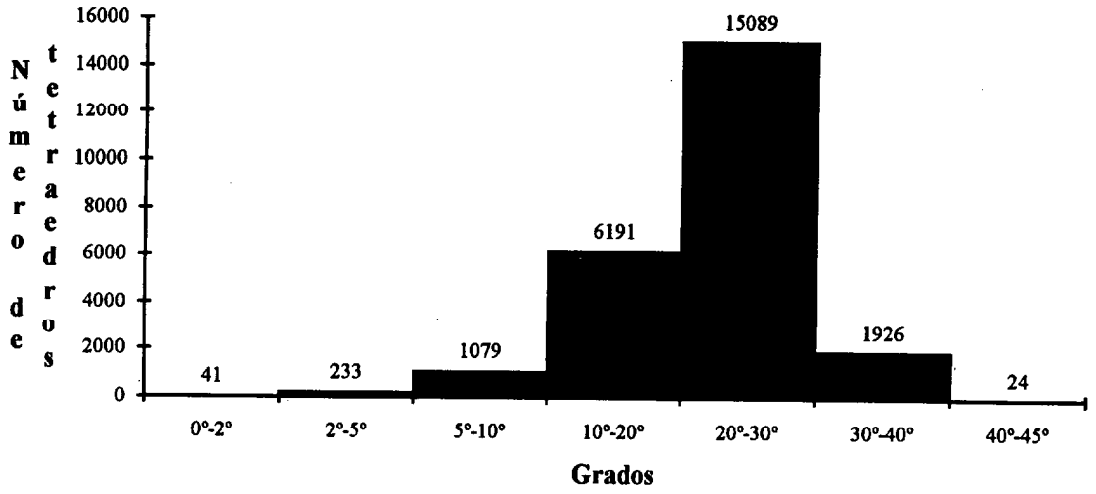


Figura 66. Medida de la calidad de la triangulación del objeto.

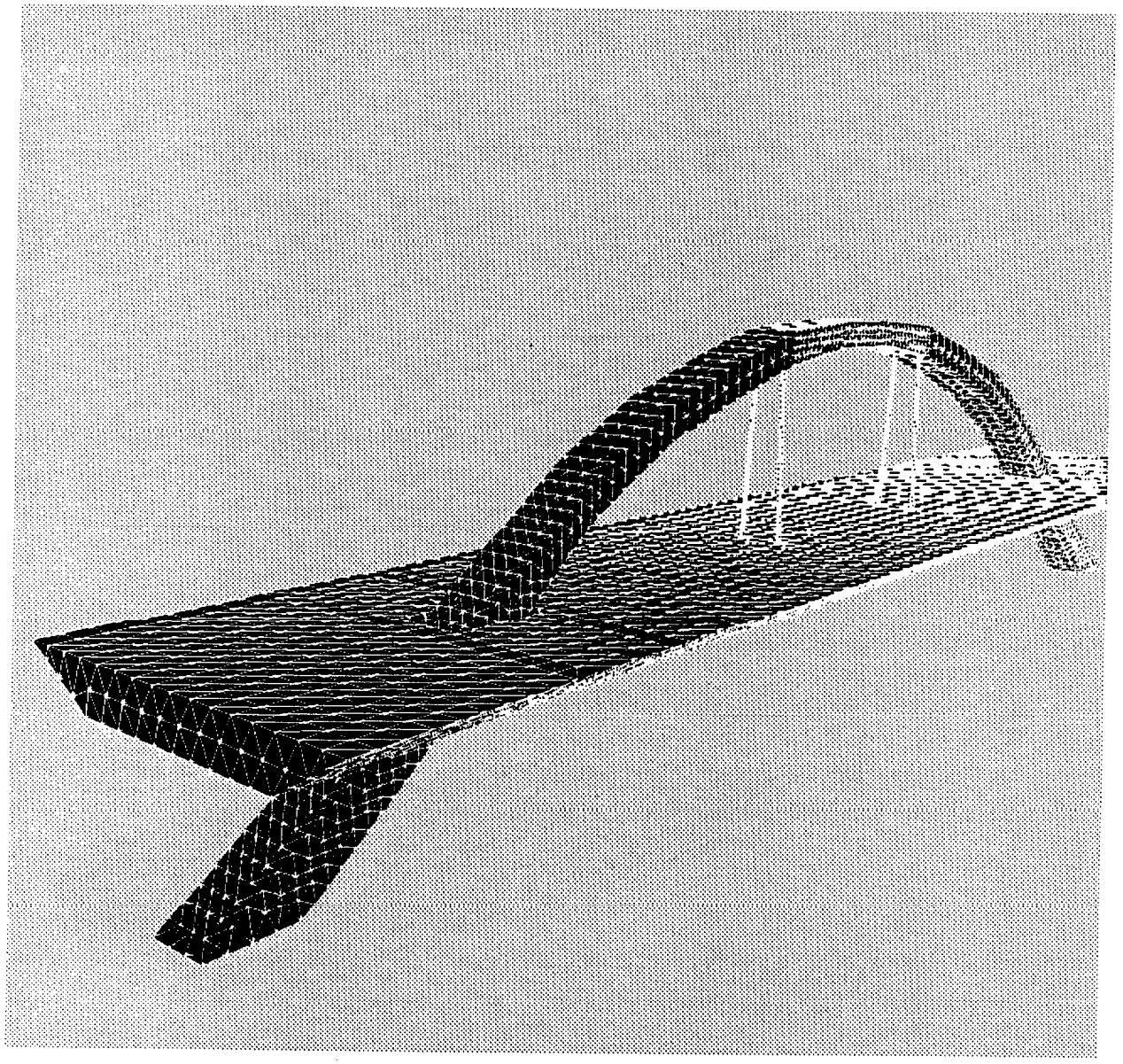


Figura 67. Triangulación del dominio.

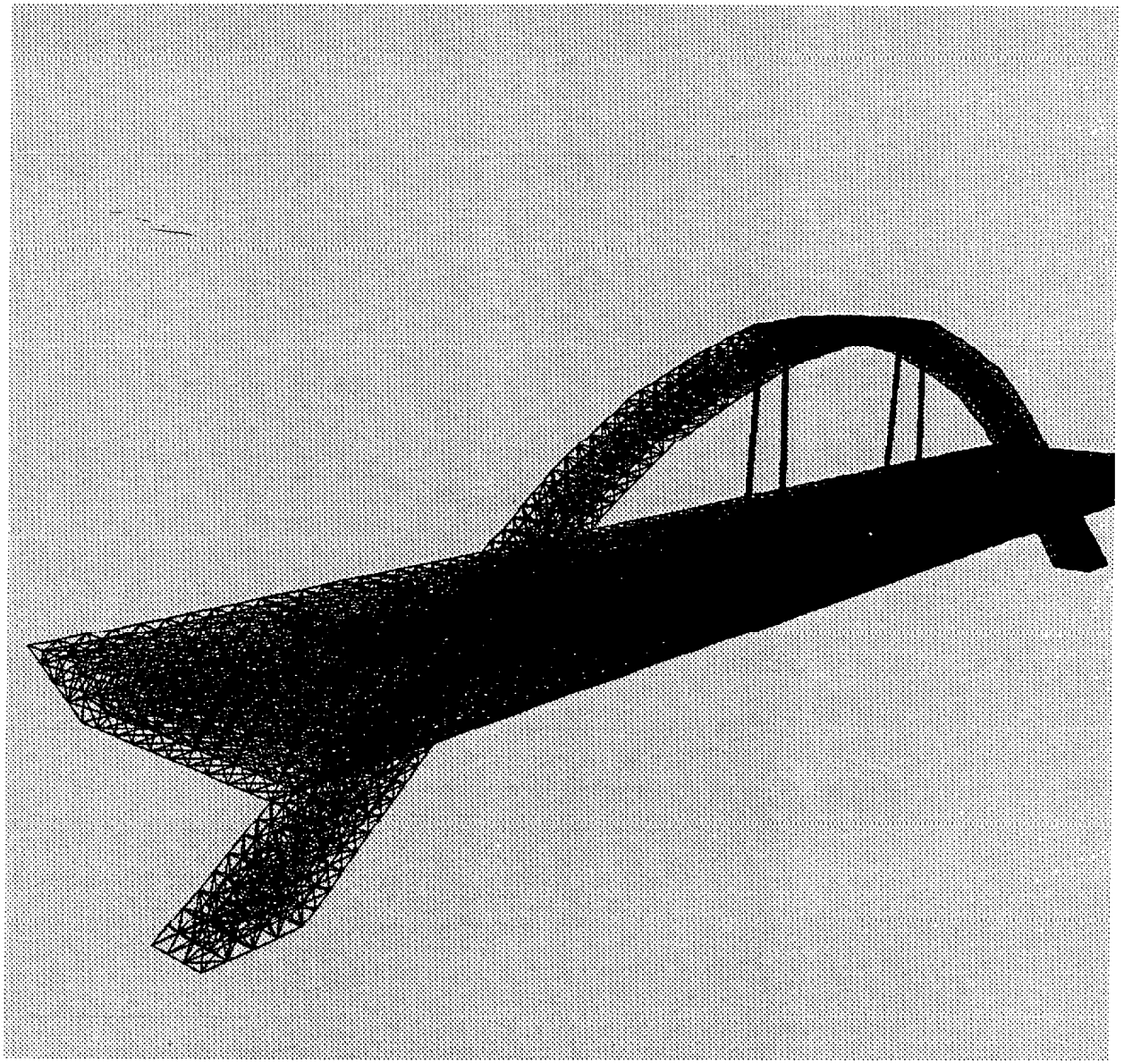


Figura 68. Líneas de la triangulación del dominio.

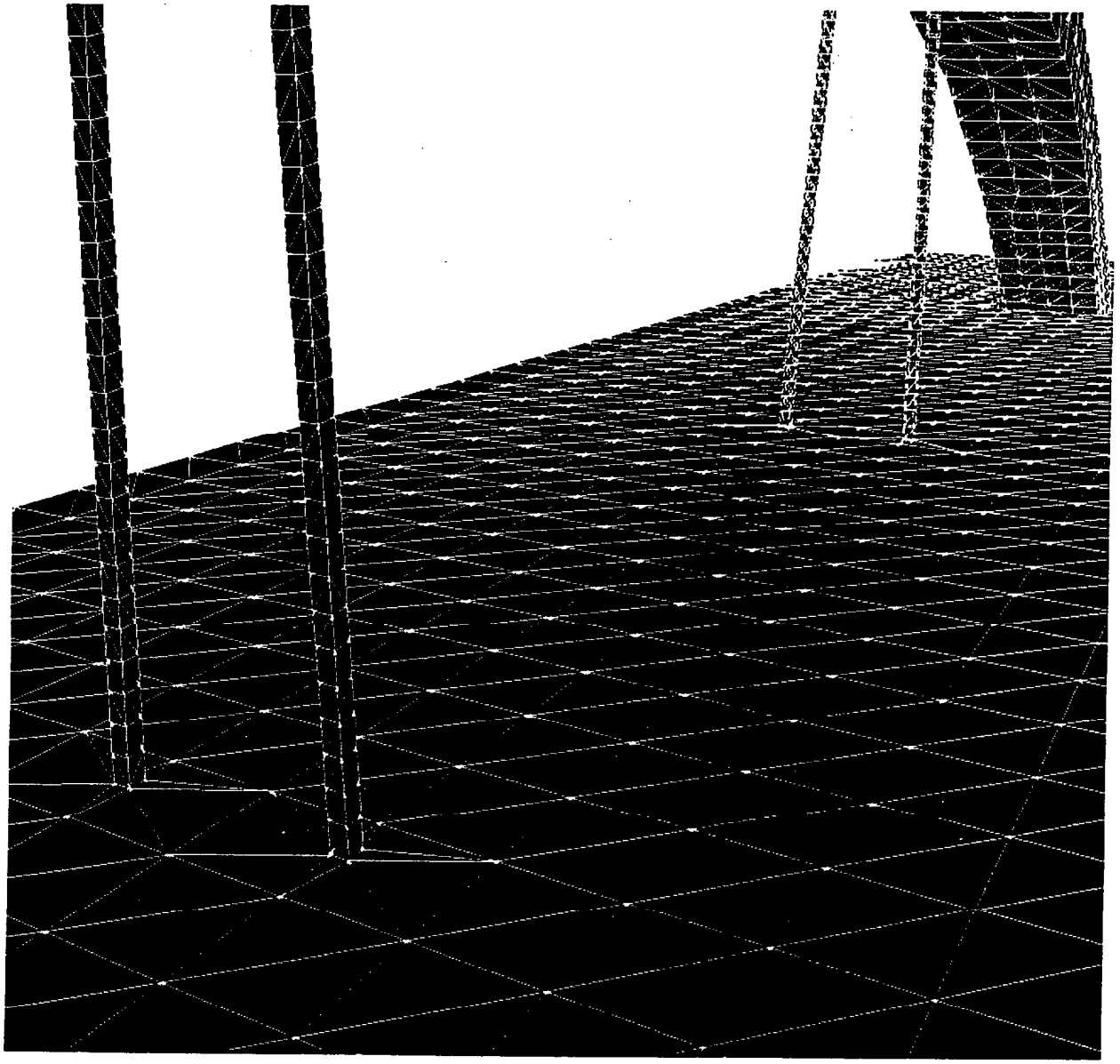


Figura 69. Detalle de los tirantes del puente.

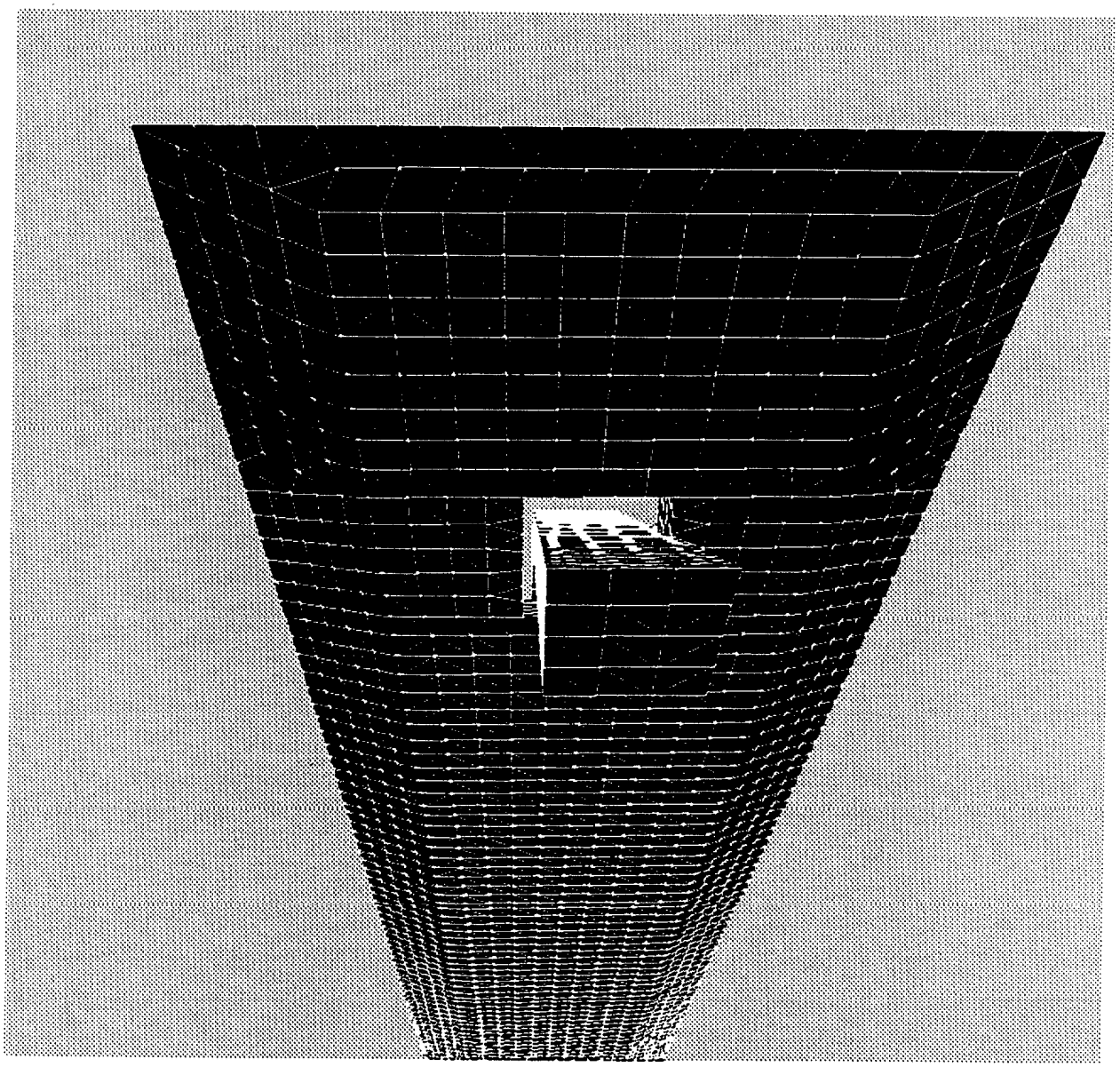


Figura 70. Vista de la parte inferior del puente y de la inserción del arco.

4.2. Límites del programa de triangulación.

Obviamente, el empleo de aritmética no exacta impone límites a la capacidad de triangular conjuntos de puntos en los que las distancias entre ellos sean muy pequeñas. Para probar este aspecto del programa hemos construido una pirámide en la que los puntos de las aristas se concentran sobre dos de sus vértices según una ley cúbica. La distancia mínima entre puntos para la que aún funcionaba el programa estaba en torno a 8×10^{-7}. Cuando se reduce esta distancia, el programa plantea problemas de diversos tipos. Por ejemplo, es incapaz de encontrar el núcleo, o no es capaz de encontrar un conjunto D_1^i ε-en forma de estrella con valores de ε superiores al límite impuesto por el ordenador, alrededor de 10^{-16} trabajando con doble precisión.

APLICACIÓN: PIRÁMIDE
Nº de puntos en el objeto: 700
Nº total de puntos en la triangulación: 770
Nº de tetraedros del objeto: 889
Nº total de tetraedros en la triangulación: 5.531

Tabla 7.

Características del procedimiento CONJUNTO D_1^i
Valor inicial de ε: 0'001
Nº máximo de iteraciones necesaria para converger sin cambiar ε: 46
Nº promedio de iteraciones necesaria para converger sin cambiar ε: 3'28571
Nº máximo de cambios $\varepsilon \rightarrow \varepsilon/5$ necesarios para converger: 5
Distancia típica del objeto (tamaño aproximado): 2
Distancia mínima entre puntos: 10^{-6}

Tabla 8.

Los datos que se aportan en las dos tablas anteriores corresponden a una distancia mínima entre puntos de 10^{-6}, por tanto, próxima al límite. Si se pretende triangular conjuntos en los que haya puntos que disten menos de esta cantidad, habrá que aumentar la precisión del ordenador.

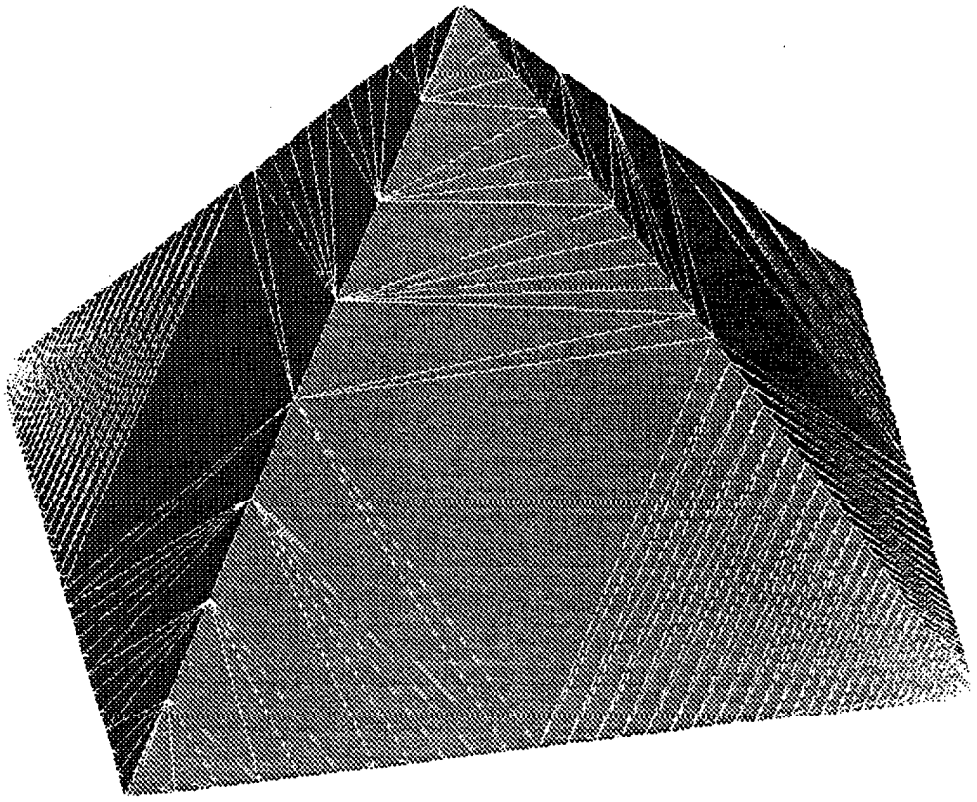


Figura 71. Acumulación de puntos cerca de dos de los vérices de la pirámide; inserción sobre las aristas.

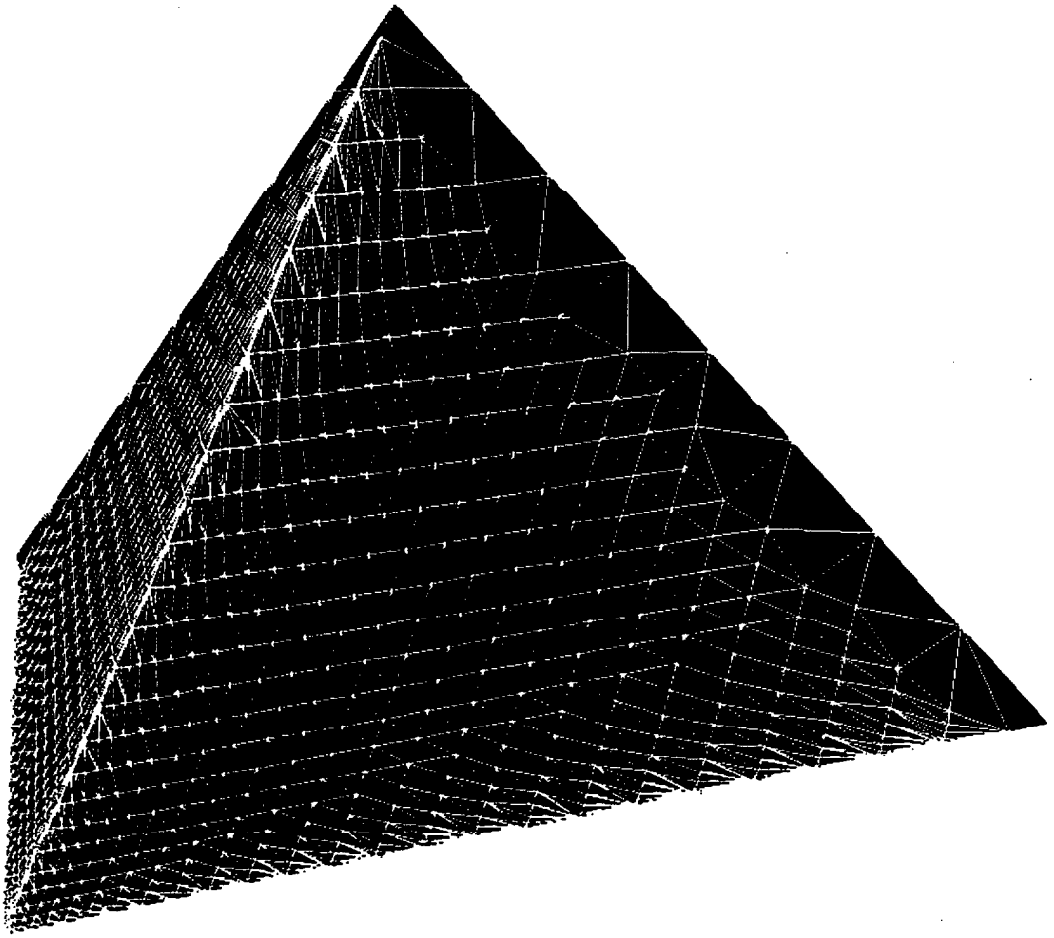


Figura 72. Acumulación de puntos cerca de dos de los vérices de la pirámide; inserción sobre las aristas y las caras.

5. CONCLUSIONES

Las conclusiones más relevantes que se pueden extraer de este trabajo son las siguientes:

1.- La triangulación de Delaunay es un procedimiento muy adecuado para la construcción de mallas para la aplicación del MEF. Hemos desarrollado de un programa en FORTRAN para la generación de mallas tridimensionales de dominios poliédricos basado en dicha triangulación. El algoritmo empleado para este propósito es una modificación del algoritmo de Watson.

2.- La posibilidad de incluir nuevos puntos en una triangulación preexistente deja abierta la posibilidad de utilizar el programa de triangulación en problemas que requieran mallas adaptativas. En otras palabras, este programa permite el refinamiento local de la triangulación siempre y cuando se disponga de un criterio adecuado de inclusión de puntos en las zonas donde los estimadores de error del MEF lo requieran.

3.- Disposiciones especiales de los puntos pueden hacer que los algoritmos que generan la triangulación de Delaunay tengan serios problemas, dando lugar a construcciones topológicamente incorrectas o tetraedros muy *degenerados* no aptos para la aplicación del MEF. En esta tesis hemos presentado un algoritmo capaz de paliar, en gran medida, estos problemas. Los límites en la actuación de este algoritmo han sido expuestos en el apartado 4.2.

4.- La calidad de la triangulación depende de la disposición espacial de los puntos que la definen. Hemos desarrollado un programa capaz de generar puntos en la superficie y en el interior de objetos poliédricos, de manera que la calidad de la triangulación sea aceptable y que ésta sea conforme, al menos en dominios con geometrías no muy complicadas, con la frontera de dicho dominio. Las densidades de puntos de las aristas, caras y regiones son especificadas en la entrada de datos.

5.- Se ha comprobado experimentalmente en bastantes ejemplos que la complejidad algorítmica del programa de triangulación es casi lineal en relación al número de puntos. Los tiempos de CPU que se necesitan para triangular un conjunto de unos 10.000 puntos, en una estación de trabajo HP-730, están en torno al minuto.

6. LÍNEAS FUTURAS DE INVESTIGACIÓN

Este trabajo deja abiertas numerosas líneas de investigación para trabajos futuros. Algunas de ellas se plantean como simples mejoras de los programas que hemos desarrollado, otras tienen objetivos más complejos y por tanto realizables a más largo plazo.

1.- Uno de los primeros objetivos que nos hemos planteado es el de aplicar el generador de mallas tridimensionales a la resolución de diversos problemas mediante el MEF.

2.- La entrada de datos del programa de generación automática de puntos resulta algo tediosa cuando se pretenden definir dominios muy complejos. Una primera mejora de este programa podría ir encaminada a modificar la información que se necesita para definir la geometría del dominio. En este sentido, sería conveniente que la entrada de datos admitiera directamente la especificación de poliedros no convexos sin necesidad de descomponerlos en convexos.

3.- En numerosas ocasiones los dominios en los que se va a aplicar el MEF tienen superficies curvas. Por el momento, la única manera en que nosotros podemos tratar estas superficies es descomponiéndolas en pequeños poliedros convexos. Las técnicas CAD permiten un tratamiento eficaz de tales superficies. Un objetivo a medio plazo sería desarrollar un programa de generación automática de puntos basado en estas técnicas.

4.- En el apartado 3.6 de esta tesis hemos indicado cuál sería una posible línea de trabajo para tratar el problema de la conformidad de la triangulación con el dominio. Los resultados obtenidos hasta el momento han sido prometedores, si bien, no exentos de conflictos. Tenemos intención de continuar investigando las posibilidades que abre esta línea de trabajo.

5.- Como se comentó en el apartado 2.4, la estructura de datos estaba pensada para tener gran versatilidad y fácil aplicación a diversos programas del MEF. Como contrapartida, tiene el inconveniente de requerir gran cantidad de memoria. Introduciendo algunas modificaciones en los algoritmos que maneja el programa de triangulación podríamos conseguir que éstos se adecuaran a una estructura de datos más ahorrativa. Además, se podría sustituir el lenguaje de programación utilizado, el FORTRAN, por un lenguaje más actual y con capacidad de dimensionamiento dinámico de vectores y matrices, como es el C.

6.- Existen algunas técnicas de reciente aparición, basadas en algoritmos genéticos, que permiten mejorar la calidad de una malla. Una posible línea de actuación se centraría en la aplicación de estos algoritmos, o de cualquier otra técnica de mejora de la calidad, a las mallas tridimensionales de tipo Delaunay.

7.- Las mallas adaptativas en 3-D se encuentran en su primera fase de desarrollo. Como ya se ha comentado con anterioridad, los algoritmos incrementales, como el que hemos utilizado, ofrecen la posibilidad de refinar localmente la malla mediante la adición de nuevos puntos. Una interesante línea de actuación sería elaborar procedimientos de inserción de nuevos puntos de manera que se consiga un adecuado refinamiento local de la malla. Se ha pensado también en la posibilidad de encontrar algoritmos que permitan quitar puntos de la malla existente, y que la reconstruyan después, para así conseguir un desrefinamiento local de la misma. Combinando los dos procedimientos anteriores se conseguiría un generador de mallas capaz de refinar o desrefinar localmente, según las necesidades del problema. Estos generadores de mallas son especialmente útiles en problemas evolutivos.

8.- Como alternativa a la estrategia anterior se ha pensado en la posibilidad de refinar las mallas generadas por nuestro programa mediante los algoritmos propuestos

por Rivara o Bänsch en [RivLev92] y [Bäns91], respectivamente. Estos algoritmos, a base de mallas encajadas tienen la ventaja de conservar la conformidad de la malla con el dominio. Además, son especialmente indicados para adaptarlos a procesos de desrefinamiento.

La resolución de diversos problemas evolutivos en 2-D utilizando algoritmos de refinamiento/desrefinamiento de mallas encajadas ya ha sido tratado por varios autores vinculados a este Departamento [FerMon94] [Plaz93].

REFERENCIAS

[AllFig92] D. Alleau, J. P. Figeac and B. Mantel, J. Periaux, "Unstructured Mesh Generation and Adaptation Techniques for 3-D Comprensible Flows", *Comp. Meth. in Appl. Sciences* 225-232 (1992).

[Bake89] J. Baker, "Developments and Trends in Three-dimensional Mesh Generation", *Appl. Num. Math.* **5**, 275-304 (1989).

[Bäns91] E. Bänsch, "Local Mesh Refinement in 2 and 3 Dimensions", *Imp. Comp. in Scien. and Eng.* **3**, 181-191 (1991).

[BerEpp92] M. Bern and D. Eppstein, "Mesh Generation and Optimal Triangulation", *Computing in Euclidean Geometry*, 23-90, World Cientific, Singapore, 1992.

[BorErd93] F. Boremann, B. Erdmann and R. Kornhuber, "Adaptative Multilevel Methods in Three Space Dimensions", *Int. J. Num. Meth. Eng.* **36**, 3187-3203 (1993).

[BovCar92] S. W. Bova and G. F. Carey, "Mesh Generation/Refinement Using Concepts and Iterated Function Systems", *Int. J. Num. Meth. Eng.* **33**, 287-305 (1992).

[Bowy81] A. Bowyer, "Computing Dirichlet Tesselations", *The Computer Journal* **24**, 162-166 (1981).

[Bura90] E. Buratynsky, "A Fully Automatic Three-dimensional Mesh Generator for Complex Geometries", *Int. J. Num. Meth. Eng.* **30**, 931-952 (1990).

[CavFie85] J. Cavendish, D. Field and W. H. Frey, "An Approach to Automatic Three-dimensional Finite Element Mesh Generation", *Int. J. Num. Meth. Eng.* **21**, 329-347 (1985).

[CiaFor89] P. Ciampolini, A. Forghieri, A. Pierantoni, A. Gnudi, M. Rudan and G. Baccarani, "Adaptative Mesh Generation Preserving the Quality of the Initial Grid", *IEEE Tran. on Com.-Aid. Desing*. **8**, 490-500 (1989).

[EscMon93] J. M. Escobar y R. Montenegro, "Introducción a la generación de mallas en 3-D", *Actas del XIII C.E.D.Y.A., III Congreso de Matemática Aplicada* (a aparecer), Madrid 13-15 Septiembre 1993.

[EscMon94a] J. M. Escobar y R. Montenegro, "Construction of 3-D Meshes Using the Triangulation of Delaunay", *VSP International Colloquium on Numerical Analysis* (a aparecer), Plovdiv, (1994).

[EscMon94b] J. M. Escobar y R. Montenegro, "Several Aspects of the Three-dimensional Triangulation of Delaunay", *Advances in Post and Preprocessing for Finite Element Technology, Civil Comp Ltd.*, Edinburgh, 193-204 (1994).

[FerMon94] L. Ferragut, R. Montenegro and A. Plaza, "Efficient refinement/ derefinement algorithm of nested meshes to solve evolution problems", *Comm. in Num. Meth. in Eng.* **10**, 403-412 (1994).

[Fort92] Fortune, "Voronoi Diagrams and Delaunay Triangulations", *Computing in Euclidean Geometry*, 193-233, World Cientific, Singapore, 1992.

[Fried93] O. Friedrich, "A New Method for Generating Inner Points of Triangulations in Two Dimensions", *Comp. Meth. in Mech. and Eng.* **104**, 77-86 (1993).

[GeoHec90] P. L. George, F. Hecht and E. Saltel, "Fully Authomatic Mesh Generator for 3-D Domains of Any Shape", *Imp. Comp. in Scien. and Eng.* **2**, 187-218 (1990).

[GeoHec91a] P. L. George, F. Hecht and E. Saltel, "Automatic Mesh Generator with Specified Boundary", *Comp. Meth. in Mech. and Eng.* **92**, 269-288 (1991).

[GeoHec91b] P. L. George, F. Hecht and M. G. Vallet, "Creation of Internal Points in Voronoi's Type Method. Control Adaptation", *Adv. Eng. Software* **13**, 303-312 (1991).

[GeoHer92] P. L. George and F. Hermeline, "Delaunay's Mesh of a Convex Polyhedron in Dimension *d*. Application to Arbitrary Polyhedra", *Int. J. Num. Meth. Eng.* **33**, 975-995 (1992).

[Geor91] P. L. George, *Automatic Mesh Generation*, John Wiley & Sons, París, 1991.

[GröLov93] M. Grötschel, L. Lovász and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, Germany., 1993.

[HanLev92] A. Hansen and P. Levin, "On Conforming Delaunay Mesh Generation", *Adv. Eng. Software* **14**, 129-135 (1992).

[HenWal93] R. F. Henry and R. A. Walters, "Geometrically Based, Automatic Generator for Irregular Triangular Networks", *Int. J. Num. Meth. Eng.* **9**, 555-566 (1993).

[JinTan93] H. Jin and I. Tanner, "Generation of Unstructured Tetrahedral Meshes by Advancing Front Technique", *Int. J. Num. Meth. Eng.* **36**, 1805-1823 (1993).

[Joe94] B. Joe, "Tetrahedral Mesh Generation in Polyhedral Regions Based on Convex Polyhedron Descompositions", *Int. J. Num. Meth. Eng.* **37**, 693-713 (1994).

[JohSul93] B. P. Johnson and J. M. Sullivan, "A Normal Ofsetting Technique for Mesh Generation in Three Dimensions", *Int. J. Num. Meth. Eng.* **36**, 1717-1734 (1993).

[KanGol91] S. Kanaganathan and B. Goldstein, "Comparations of Four-point Adding Algorithms for Delaunay-type Three-dimensional Mesh Generators", *IEEE Trans. on Mag.* **27**, 3444-3451 (1991).

[KasOka92] K. Kashiyama and T. Okada, "Authomatic Mesh Generation Method for Shallow Water Flow Analysis", *Int. J. Num. Meth. in Flu.* **15**, 1037-1057 (1992).

[KnuSte93] P. Knupp and S. Steinberg, *Fundamentals of Grid Generation*, CRC, U.S.A., 1993.

[LöhBau92] R. Löhner and J. D. Baum, "Adaptative *h*-Refinement on 3D Unstructured Grids for Transient Problems", *Int. J. Num. Meth. Eng.* **14**, 1407-1419 (1992).

[LöhCam92] R. Löhner, J. Camberos and Marshall Merrian, "Parallel Unestructured Grid Generation", *Comp. Meth. in Mech. and Eng.* **95**, 343-357 (1992).

[MarMer92] D. Marshall and L. Merriam, "Parallel Implementation of an Algorithm for Delaunay Triangulation", *Com. Flu. Dynam.* **2**, 907-912 (1992).

[MonPlaz95] R. Montenegro, A. Plaza, J. M. Escobar y L. Ferragut, "Aspects abaut Mesh Generation for Finite Elements Method", *Neural, Parallel and Scientific Computation.* **1**, (a aparecer), Atlanta 1995.

[Pach93] J. Pach, *New Trends in Discrete and Computational Geometry*, Springer-Verlag, U.S.A., 1993.

[Perr88] A. Perronnet, "Un algorithme de tétraèdrisation d'un objet muti-matériaux ou de léxtérieur d'un objet", Laboratoire d'Analyse Numérique. Université Pierre et Marie Curie, Centre National de la Recherche Scientifique, 75252 Paris Cedex-05 France, 1988.

[Plaz93] Plaza A., "Algoritmos de desrefinamiento en mallados estructurados bidimensionales", *Tesis Doctoral*, Departamento de Matemáticas, Universidad de Las Palmas de Gran Canaria, 1993.

[PouRad91] M. Pourazady and M. Radhakrishman, "Optimization of Triangular Mesh", *Com. and Struc.* **40**, 795-804 (1991).

[PreSha85] F. P. Preparata and M. I. Shamos, *Computational Geometry. An Introduction*, Springer-Verlag, New York, 1985.

[RanSch93] E. Rank, M. Schweingruber and M. Sommer, "Adaptative Generation and Transformation of Quadrilateral Meshes", *Comm. in Num. Meth. in Eng.* **9**, 121-129 (1993).

[Reba93] S. Rebay, "Efficient Mesh Generation by Means of Delaunay Triangulation and Bowyer-Watson Algorithm", *J. of Com. Phy.* **106**, 125-138 (1993).

[RivLev92] M. C. Rivara and C. Levin, "A 3-D Refinement Algorithm Suitable for Adaptative and Multi-Grid Techniques", *Comm. in Appl. Num. Meth.* **8**, 281-290 (1992).

[SchRan93] M. Schweingruber and E. Rank, "Adaptative Mesh Generation for Triangular or Quadrilateral Elements", *Num. Meth. in Eng.* 9-15 (1993).

[SchShe90] W. J. Schroeder and M. S. Shephard, "A Combined Octree/Delaunay Method for Fully Automatic 3-D Mesh Generation", *Int. J. Num. Meth. Eng.* **29**, 37-55 (1990).

[SloHou84] S. W. Sloan and G. T. Houlsby, "An implementeision of Watson's Algorithm for Computing 2-Dimensional Triangulations", *Adv. Eng. Software* **6**, 192-197 (1984).

[SouGat93] L. T. Souza and M. Gattass, "A New Scheme for Mesh Generation and Refinement Using Graph Theory", *Com. and Struc.* **46**, 1073-1084 (1993).

[TanHol92] T. Taniguchi, K. Holz and C. Ohta, "Grid Generation for 2D Flow Problems", *Int. J. Num. Meth. in Flu.* **15**, 985-997 (1992).

[VilHän93] R. Vilsmeier and D. Hänel, "Adaptative Methods on Unestructured Grids for Euler and Navier-Estokes Equations", *Computers Fluids* **22**, 485-499 (1993).

[Wats81] D. F. Watson, "Computing the *n*-dimensional Delaunay Tessellation with Application to Voronoy Polytopes", *The Computer Journal* **24**, 167-172 (1981).

[WeaHas92] N. P. Weatherhill and O. Hassan, "Efficient Three-Dimensional Grid Generation Using the Delaunay Triangulation", *Com. Flu. Dynam.* **2**, 961-968 (1992).

[WeaHas94] N. P. Weatherhill and O. Hassan, "Efficient Three-Dimensional Delaunay Triangulation with Automatic Point Creation and Imposed Boundary Constraints", *Int. J. Num. Meth. Eng.* **37**, 2005-2039 (1994).

[Weath92] N. P. Weatherhill, "The Integrity of Geometrical Boundaries in the Two-dimensional Delaunay Triangulation", *Comm. in Appl. Num. Meth.* **6**, 101-109 (1992).

[WrigJac94] J. P. Wright and A. G. Jack, "Aspects of Three-Dimensional Constrained Delaunay Meshing", *Int. J. Num. Meth. Eng.* **37**, 1841-1861 (1994).

[YerShe84] M. A. Yerry and M. Shephard, "Automatic Three-dimensional Mesh Generation by Modified-octree Technique", *Int. J. Num. Meth. Eng.* **20**, 1965-1990 (1984).

[ZhoSha89] J. M. Zhou, K. R. Shao and J. D. Lavers, "Automatic Mesh Generation Allowing for Efficient a Priori and a Posteriori of the Number and Distribution of Elements", *IEEE Trans.. on Mag.* **25**, 2983-2985 (1989).

[ZhoZho92] J. M. Zhou, K. D. Zhou and R. Shao, "Automatic Generation of 3D Meshes for Complicated Solids", *IEEE Trans.. on Mag.* **28**, 1759-1762 (1992).