UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

DEPARTAMENTO DE MATEMÁTICAS

TESIS DOCTORAL

ESTRUCTURAS DE DATOS TIPO GRAFO EN LOS ALGORITMOS DE REFINAMIENTO Y DESREFINAMIENTO BASADOS EN LA BISECCIÓN POR EL LADO MAYOR. APLICACIONES

JOSÉ PABLO SUÁREZ RIVERO

Las Palmas de Gran Canaria, 2001

Título de la tesis:

ESTRUCTURAS DE DATOS TIPO GRAFO EN LOS ALGORITMOS DE REFINAMIENTO Y DESREFINAMIENTO BASADOS EN LA BISECCIÓN POR EL LADO MAYOR. APLICACIONES

Thesis title:

GRAPH-BASED DATA STRUTURES FOR REFINEMENT AND DEREFINEMENT ALGORITHMS BASED ON THE LONGEST EDGE BISECTION. APPLICATIONS

GRAPH-BASED DATA STRUTURES FOR REFINEMENT/DEREFINEMENT ALGORITHMS BASED ON THE LONGEST EDGE BISECTION. APPLICATIONS.

ABSTRACT

In the last years, mesh generation algorithms have been largely investigated. These algorithms are basic tools in numerical methods, as in the finite element method or in the piecewise linear approximation to a given function. The algorithms are aslo relevant in graphic engineering, computer graphics, surfaces and solid modeling, industrial design, visualization, CAD/CAM, etc. Furthermore, in an increasing way, the algorithms are getting used in Science and Technology Numerical Simulations; in this wide topic it is specially critical to provide algorithms with efficient perfomance in terms of time and space.

In the refinement and coarsening algorithms, an important point is to have a suitable data structure which makes it possible that the algorithms perform in an efficient way. This is due to the complexity of operations in the elements partitions which turn in a critical issue when the dimension increases and the number of elements grow rapidly.

This thesis investigates a new class of data structures which are based on the graph theory. The data structures are used to design new efficient versions of refinement and coarsening algorithms. We explore the main properties of the data structures in 2D. Moreover, a number of applications using the new version of the algorithms are provided.

Universidad de Las Palmas de Gran Canaria

Departamento de Matemáticas
Departamento de Cartografía y Expresión Gráfica en la Ingeniería

TESIS DOCTORAL

ESTRUCTURAS DE DATOS TIPO GRAFO EN LOS ALGORITMOS DE REFINAMIENTO Y DESREFINAMIENTO BASADOS EN LA BISECCIÓN POR EL LADO MAYOR. APLICACIONES.

José Pablo Suárez Rivero

Octubre de 2001

Tesis Doctoral:

ESTRUCTURAS DE DATOS TIPO GRAFO EN LOS ALGORITMOS DE REFINAMIENTO Y DESREFINAMIENTO BASADOS EN LA BISECCIÓN POR EL LADO MAYOR. APLICACIONES.

Programa de Doctorado:

Simulación Numérica en Ciencia y Tecnología

Bienio:

1997-1999

Departamento:

Departamento de Matemáticas

Universidad:

Universidad de Las Palmas de Gran Canaria

Las Palmas de Gran Canaria, a de de 2001

Autor	Director
Jose Pablo Suárez Rivero	Dr. Ángel Plaza de La Hoz

Agradecimientos

Quisiera agradecer a mi director de tesis Dr. Ángel Plaza por la dedicación e interés que ha mostrado desde el primer momento en toda mi actividad investigadora. Sus acertadas sugerencias y confianza depositada en mí han hecho posible convertir en realidad un proyecto de tesis como el realizado. La sintonía en nuestro trabajo de investigación sigue en la actualidad. Asimismo, al profesor Dr. Graham F. Carey quien en mi última estancia en el TICAM (Texas Institute for Computational and Applied Mathematics) de la Universidad de Texas, mostró un gran interés en mi trabajo de investigación que lo ha fortalecido. Los densos meses de trabajo bajo su dirección han sido claves para el término de la tesis. A la profesora María-Cecilia Rivara de la Universidad de Chile quien también siempre ha estado colaborando y aportando con su experiencia y saber numerosas sugerencias e ideas contempladas en esta tesis.

Resumen

Los algoritmos de generación de mallas han sido objeto de numerosas investigaciones en los últimos años. Estos algoritmos constituyen herramientas básicas en los métodos numéricos, como en el método de elementos finitos o en la aproximación lineal a trozos de funciones, así como en la industria gráfica, como gráficos por ordenador, diseño y modelado de sólidos y superficies, visualización, CAD/CAM, etc. Cada vez más se utilizan en entornos de **Simulación Numérica en Ciencia y Tecnología**, en donde es crítico que los algoritmos sean eficientes en términos de tiempo y espacio de cómputo.

En los algoritmos de refinamiento y desrefinamiento de mallas es crucial disponer de una buena estructura de datos que haga eficiente la ejecución de los mismos. Ello se debe a las complejas operaciones que surgen en la partición de elementos, lo cual se acentúa cuando la dimensión del problema aumenta y la cantididad de elementos crece rápidamente.

Esta tesis investiga una clase de estructuras de datos basadas en grafos que permite diseñar algoritmos eficientes de refinamiento y desrefinamiento de mallas y propone algunas aplicaciones.

El trabajo está estructurado como sigue: en el capítulo primero se hace una introducción general de la generación de mallas, incluyendo ideas, conceptos, algoritmos y técnicas y aplicaciones. Asimismo, se presentan algunas ideas básicas de diseño de estructuras de datos. Se concluye el capítulo señalando los objetivos propuestos en esta tesis.

El capítulo segundo se dedica al estudio de las particiones geométricas de elementos simpliciales en 2D y 3D y de los algoritmos de refinamiento y desrefinamiento más importantes de la literatura, especialmente los basados en el esqueleto.

El capítulo tres hace una revisión de las estructuras de datos geométricas y espaciales más importantes en la representación de mallas.

El capítulo cuatro presenta una nueva estructura de datos basada en grafos que de forma natural se ajusta a los algoritmos de refinamiento y desrefinamiento basados en el esqueleto de Plaza y Carey. Se presentan las propiedades matemáticas y computacionales de dicha estructura de datos. Asimismo, se proporcionan versiones nuevas de los algoritmos de refinamiento en dimensión dos y tres y del algoritmo de desrefinamiento en dimensión dos. Finalmente, se formulan y demuestran diversas propiedades de la partición en cuatro triángulos por el lado mayor.

En el capítulo quinto se presentan aplicaciones: en la sección primera, modelos digitales del terreno y generalización de terrenos; en la segunda sección, niveles de detalle en gráficos por ordenador y $VRML$; y finalmente, en la sección tercera se incorpora el algoritmo de refinamiento en 2D a un código de elementos finitos comercial y se resuelve un problema elíptico no lineal.

El capítulo seis se destina a presentar las conclusiones resultantes de esta tesis y se proponen diversas líneas de trabajo futuro.

Finalmente, se presenta un anexo que recoge las tres publicaciones más importantes que han dado lugar las investigaciones llevadas a cabo en la tesis.

Abstract

In the last years, mesh generation algorithms have been largely investigated. These algorithms are basic tools in numerical methods, as in the finite element method or in the pieceway linear approximation to a given function. The algorithms are also relevant in the graphic industry, as in computer graphics, surfaces and solids modeling and design, visualization, CAD/CAM, etc. Furthermore, in an increasing way, the algorithms are getting used in **Science and Technology Numerical Simulation**; in this wide topic it is specially critical to provide algorithms with efficient perfomance in terms of time and space.

In the refinement and coarsening algorithms, an important point is to have a suitable data structure which makes it possible that the algorithms perform in an effcicient way. This is due to the complexity of operations in the elements partitions which turn in a critical issue when the dimension increases and the number of elements grows rapidly.

The present thesis investigates a new class of data structures which are based on the graph theory. The data structures are used to design new efficient versions of refinement and coarsening algorithms and applications.

The outline of the treatment is as follows: chapter one introduces mesh generation, including basic ideas, concepts, algorithms and techniques, and applications. Some preliminaries about data structures are given and the goals of the thesis close the chapter.

Second chapter studies the geometric partitions for simplicial elements in both 2D and 3D. It is also discussed the refinement and coarsening algorithms which are more relevant in the literature, specially the algorithms based on the skeleton.

In the third chapter the more important geometric and spatial data structures in the representation of meshes are reviewed.

Fourth chapter introduces a new graphs based data strucuture which gives a natural comprehension to the skeleton based refinement and coarsening algorithms by Plaza and Carey. It is presented some mathematical and computational prooperties related to the new data structures. Besides, it is provided new and efficient versions for the refinement algorithms in 2D and 3D and for the coarsening algorithm in 2D. Finally, we present several properties of the longest-edge four triangles partition and the respective formal proofs.

Some applications are given in the fifth chapter: in section one, digital elevation models and terrains generalization; in section two, levels of detail in computer graphics and $VRML$; finally, section three solves a 2D non linear eliptic problem using a commercial finite element package in which the new refinement algorithm is used.

In the sixth chapter some remarkable conclusions derived from the thesis are emphasized, and some possible points of research for the future are pointed out.

Finally, it is enclosed the three main publications based on this thesis.

Índice general

1. Introducción y Objetivos **1**
1.1. Algoritmos de generación de mallas 3
1.1.1. Quadtree-octree 4
1.1.2. Delaunay 7
1.1.3. Avance frontal 16
1.2. El Método de los Elementos Finitos 18
1.2.1. Formulación del problema 20
1.2.2. El problema del refinamiento local 22
1.3. Representación gráfica en 2D y 3D 25
1.4. Estrategias de refinamiento 28
1.5. Estructuras de datos geométricas 31
1.6. Aportación y objetivos de esta tesis 35

2. Particiones y algoritmos **37**
2.1. Definiciones y preliminares 37
2.2. Algoritmos de refinamiento y particiones en 2D 43
2.3. Algoritmos de refinamiento y particiones en 3D 47
2.4. Las particiones 4T-LE y 8T-LE 53
2.5. Los algoritmos 2D-SBR y 3D-SBR 61
2.6. Comportamiento asintótico de las particiones 65
2.6.1. Una medida de irregularidad topológica 65
2.6.2. Comportamiento asintótico de la media del grado del nodo 66
2.6.3. Funciones de generación 68

3. Estructuras de datos geométricas **75**
3.1. Estructuras de datos en la representación de mallas 75
3.1.1. Relaciones de adyacencia completas 79
3.1.2. Lista de aristas doblemente conectada (DCEL) 80

3.1.3. Estructura de datos Winged-Edge 82
3.1.4. Estructura de datos Half-Edge 85
3.1.5. Estructura de datos Quad-Edge 89
3.2. Estructuras de datos espaciales 93
3.2.1. Acceso Indexado 94
3.2.2. Acceso Hash 95
3.2.3. Integración arbórea 97

4. Estructuras de datos tipo grafo 103
4.1. Definiciones y preliminares 103
4.2. El grafo basado en el esqueleto 106
4.2.1. El grafo del 1-esqueleto en 2D 111
4.2.2. El grafo del 1- y 2-esqueleto en 3D 113
4.3. Implementación del grafo basado en el esqueleto 118
4.4. Algoritmos 2D-SBR y 3D-SBR 120
4.5. Algoritmo de desrefinamiento 2D-SBD 127
4.6. Propiedades geométricas de la partición 4T-LE 133
4.6.1. Definiciones y preliminares 136
4.6.2. Propiedades geométricas de la partición 4T-LE 137
4.6.3. Experimentos con el algoritmo 2D-SBR 147

5. Aplicaciones 159
5.1. Introducción 159
5.2. Modelos Digitales del Terreno 160
5.2.1. Introducción 160
5.2.2. Generalización de terrenos 162
5.3. Niveles de Detalle 186
5.4. Método de Elementos Finitos 195
5.4.1. Solución de un problema elíptico no lineal 199

6. Conclusiones y Trabajo Futuro 205
6.1. Conclusiones 205
6.2. Trabajo futuro 207

A. Publicaciones 209
A.1. 7th International Meshing Roundtable'98 211
A.2. TICAM Technical Report 223

A.3. Communications in Numerical Methods in Engineering 237

Bibliografía **247**

Índice de figuras

1.1. Estructura de datos tipo quadtree 4
1.2. División de un cubo por octree 6
1.3. Conjunto de polígonos de Voronoi asociado al conjunto de puntos 8
1.4. Triangulación de polígonos de Voronoi 8
1.5. Dominio con agujeros en su interior 9
1.6. Triangulación de Delaunay después de los pasos 1 y 2 11
1.7. Triangulación de Delaunay realizado el paso 3 11
1.8. Triangulación de Delaunay 12
1.9. Formación de un cuadrilátero 12
1.10. Triangulación de Delaunay, no única 13
1.11. Inserción de un nuevo punto 13
1.12. Formación de un polígono de inserción 14
1.13. Nueva triangulación de Delaunay 14
1.14. Elección de un nodo cuando el frente avanza 16
1.15. Determinación de puntos equiespaciados 17
1.16. Triangulación de la superficie de Gran Canaria 26
1.17. Aproximación de una superficie curva en 3D. Figura tomada de [63] 27
1.18. Triangulación Delaunay del terreno 28
1.19. Diagrama de flujo de un refinamiento adaptativo 29
1.20. Refinamiento en la detección de bordes 32

2.1. Nodo no conforme en dos y tres dimensiones 39
2.2. Envoltura relativa a la teselación 40
2.3. Bisección de un símplice: casos bi y tridimensional 41
2.4. Relaciones de adyacencia posibles para las entidades de vértices (V), caras (F) y aristas (E) 42
2.5. Diversas formas de dividir un triángulo 44

2.6. Las cuatro posibilidades de división 4T 44
2.7. "*Green division*" para alcanzar la conformidad 45
2.8. Partición similar en 3D y vista coplanaria de las caras 47
2.9. Caso ilustrativo de la partición 8T-LE 48
2.10. Partición baricéntrica 3D 49
2.11. Patrones usados por Bey que cubren $(4 + 6 + 12 + 3) = 25$ posibilidades 49
2.12. Tetraedros abiertos clasificados como rojo y negro 50
2.13. Tetraedros tipo rojo y negro 51
2.14. Clasificación de los tetraedros según Liu y Joe 53
2.15. Patrones del refinamiento 4-T 54
2.16. Ejemplo del algoritmo de refinamiento 4-T 55
2.17. La división 4T-LE y su comportamiento fractal. La figura (c) muestra la molécula estable de P 57
2.18. Posición relativa de la arista mayor (numerada 1) y de las aristas secundarias (numeradas 2) de t 58
2.19. Partición de las dos caras que comparten la arista mayor. 58
2.20. Posibles divisiones 8T-LE distintas de t 59
2.21. Ejemplo del algoritmo de refinamiento 2D basado en el esqueleto para refinar t 64
2.22. Distribución de vértices en % vs. número de tetraedros por vértice. Tetraedro casi-regular 73
2.23. Distribución de vértices en % vs. número de tetraedros por vértice. Tetraedro casi plano 73

3.1. Ejemplos de mallas en 2D y 3D 78
3.2. Tipos de adyacencia completas 80
3.3. Diagrama de la DCEL 81
3.4. Esquema de almacenamiento Winged-Edge 83
3.5. Ejemplo de almacenamiento Winged-Edge 83
3.6. Esquema de la estructura de datos Half-Edge 86
3.7. Ejemplo de representación simple Half-Edge 87
3.8. Estructura de datos Half-Edge 87
3.9. Estructura de datos Split-Edge 89
3.10. Estructura de datos Quad-Edge 90
3.11. Grafo planar y su dual 90

3.12. Esquema de acceso indexado 95
3.13. Esquema de acceso Hash 96
3.14. Árbol Cuaternario 99
3.15. Árbol Kd de dos claves 101

4.1. Grafo de incidencia para un tetraedro 105
4.2. Representación gráfica del esqueleto 106
4.3. Configuraciones posibles para el grafo del $(n-1)$-esqueleto . . . 109
4.4. Grafo del esqueleto y vector T 110
4.5. Grafo del 1-esqueleto en 2D 112
4.6. Existencia de bucles en el grafo del 1-esqueleto 112
4.7. Patrones del G^1 para los tres tipos de tetraedros 114
4.8. Configuraciones posibles de G^1 para los tres tipos de tetraedros 115
4.9. 2-esqueleto para un tetraedro 116
4.10. Componentes con nodo común de G_1^2 y G_2^2 118
4.11. Niveles de refinamiento 1 y 2 del ejemplo test 124
4.12. Nivel de refinamiento 3 del ejemplo test 125
4.13. Ejemplo de refinamiento 129
4.14. La condición de desrefinamiento 131
4.15. Mallas 1 y 2 del test del algoritmo 2D-SBD 134
4.16. Mallas 3 y 4 del test del algoritmo 2D-SBD 135
4.17. (a) Malla equilibrada, (b) malla semi-equilibrada 137
4.18. Extensión por conformidad y LEEP de triángulos vecinos 139
4.19. 4T-LE en un triángulo acutángulo 141
4.20. (a) 4T-LE en un triángulo obtusángulo (b) Partición 4T-LE . . 142
4.21. (a) Triángulos terminales estables (b) Partición 4T-LE 146
4.22. Mallas de los problemas test 148
4.23. Refinamiento global 4T-LE y evolución de M1 y M2. Malla Delaunay 151
4.24. Refinamiento global 4T-LE y evolución de M1 y M2. Malla pentagonal 152
4.25. Grado de equilibrio. Malla pentagonal y Malla Delaunay 153
4.26. Malla inicial tipo Delaunay y nivel de refinamiento uno. Refinamiento global 4T-LE 154
4.27. Malla Delaunay. Niveles de refinamiento dos y tres. Refinamiento global 4T-LE 155

4.28. Malla pentagonal. Refinamiento global 4T-LE. Malla inicial y nivel de refinamiento uno 156
4.29. Malla pentagonal. Refinamiento global 4T-LE. Nivel de refinamiento dos y cuatro 157

5.1. Representación inicial del terreno de Agaete 165
5.2. Malla 2D y curvas de nivel del terreno de Agaete 166
5.3. Representación con error de 1 metro del terreno de Agaete . . . 167
5.4. Malla 2D y curvas de nivel con error de 1 metro del terreno de Agaete 168
5.5. Representación con error de 6 metros del terreno de Agaete . . . 169
5.6. Malla 2D y curvas de nivel con error de 6 metros del terreno de Agaete 170
5.7. Representación con error de 15 metros del terreno de Agaete . . 171
5.8. Malla 2D y curvas de nivel con error de 15 metros del terreno de Agaete 172
5.9. Representación con error de 36 metros del terreno de Agaete . . 173
5.10. Malla 2D y curvas de nivel con error de 36 metros del terreno de Agaete 174
5.11. Gráfica del error en metros frente al número de nodos 175
5.12. Gráfica del SNR en decibelios frente al número de nodos 175
5.13. Tiempo de obtención de las mallas frente al nivel de error 175
5.14. Representación inicial del terreno de Gáldar 176
5.15. Representación con error de 10 metros del terreno de Gáldar . . 177
5.16. Malla 2D y curvas de nivel con error de 10 metros del terreno de Gáldar 178
5.17. Representación con error de 15 metros del terreno de Gáldar . . 179
5.18. Malla 2D y curvas de nivel con error de 15 metros del terreno de Gáldar 180
5.19. Representación con error de 40 metros del terreno de Gáldar . . 181
5.20. Malla 2D y curvas de nivel con error de 40 metros del terreno de Gáldar 182
5.21. Representación con error de 90 metros del terreno de Gáldar . . 183
5.22. Malla 2D y curvas de nivel con error de 90 metros del terreno de Gáldar 184
5.23. Gráfica del error en metros frente al número de nodos 185

5.24. Gráfica del SNR en decibelios frente al número de nodos 185
5.25. Tiempo de obtención de las mallas frente al nivel de error 185
5.26. Puntos de vistas y niveles de detalle 190
5.27. Malla 2D y representación VRML con nivel de detalle 1 191
5.28. Malla 2D y representación VRML con nivel de detalle 2 (10 % error) 192
5.29. Malla 2D y representación VRML con nivel de detalle 3 (30 % error) 193
5.30. Malla 2D y representación VRML con nivel de detalle 4 (60 % error) 194
5.31. Diagrama en bloques de PDE Toolbox 196
5.32. Solución inicial de la ecuación elíptica no lineal 200
5.33. Solución en el nivel de refinamiento 1 201
5.34. Solución en el nivel de refinamiento 2 202
5.35. Solución en el nivel de refinamiento 3 203
5.36. Solución en el nivel de refinamiento 4 204

Índice de cuadros

2.1. Prueba numérica para la triangulación similar 4T 72
2.2. Medidas estadísticas I 74
2.3. Medidas estadísticas II 74

3.1. Dispositivos de almacenamiento 94

4.1. Datos de la prueba 1 123
4.2. Secuencia de triángulos distintos obtenidos por la aplicación iterativa de la partición 4T-LE 144
4.3. Malla Delaunay. Métricas M1 y M2 151
4.4. Malla pentagonal. Métricas M1 y M2 152
4.5. Malla Delaunay. Grado de equilibrio 153
4.6. Malla Pentagonal. Grado de equilibio 153
4.7. Malla Delaunay. Medida de ángulos 158
4.8. Malla Pentagonal. Medida de ángulos 158

5.1. Niveles de detalle para el terreno de Gáldar 190

Capítulo 1

Introducción y Objetivos

La generación de mallas es el proceso de discretización de un dominio físico en subdominios más pequeños (elementos). Las mallas se encuentran en diversas áreas del análisis numérico dentro de la Ciencia e Ingeniería aplicada. Ejemplos de éstas son: ajuste de curvas y superficies en la teoría de aproximación, integración numérica, diferencias finitas, volúmenes finitos y elementos finitos. Todas tienen en común la solución numérica de ecuaciones diferenciales en derivadas parciales, cuyo proceso en general puede ser descrito de la siguiente forma:

1. Definición inicial del dominio proporcionando algunos puntos.
2. Generación de la malla y etiquetado de nodos.
3. Solución numérica del problema en el dominio discretizado.
4. Refinamiento o redistribución de la malla y salto al paso 3 para mejorar la solución.
5. Presentación de la malla y solución (visualización).

No obstante, la generación de mallas se ha convertido actualmente en un tema interdisciplinar que aparece en muchas otras áreas. Citamos por ejemplo la industria gráfica, y más concretamente, gráficos por ordenador, diseño y modelado de sólidos y superficies, visualización, CAD/CAM, etc. Asimismo la geometría y la topología computacional hacen obligada visita al tema de la generación de mallas. Otras áreas más aplicadas como cartografía por ordenador y sistemas de información geográficos (GIS) han venido a recurrir a

técnicas de mallado y afines para la representación y visualización de datos espaciales.

Fundamentalmente, los dominios 2D pueden ser divididos en triángulos o cuadriláteros mientras que los dominios 3D en tetraedros o hexaedros. Idealmente la forma y distribución de elementos en el dominio son realizados por los algoritmos de generación de mallas, automáticos o no automáticos. Serán automáticos, cuando la generación de mallas sólo requiere de unos pocos parámetros iniciales y a continuación se realiza el proceso, no requiriendo la participación del usuario más que al inicio. Serán no automáticos en caso contrario, es decir, el proceso de la generación de la malla requiere de decisiones y parámetros externos.

En las décadas recientes, el método de elementos finitos ha llegado a ser un soporte fundamental en el diseño y análisis de aplicaciones industriales. Cada vez más, diseños complejos y de gran magnitud están siendo simulados usando el método de elementos finitos, lo que ha hecho aumentar la popularidad e interés por los algoritmos de mallado automático.

El problema de la generación automática de mallas considera la mejor descripción de un dominio geométrico, sujeto a un conjunto dado de nodos iniciales, especificación de tamaños para los elementos y criterios de la forma de los elementos. La geometría del dominio se compone habitualmente de vértices, curvas, superficies y sólidos, que en el mejor de los casos se describen por paquetes de CAD o de modelado de sólidos.

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 obtenidas por una transformación de coordenadas, de un cuadrado en dimensión dos ó un cubo en dimensión tres, son ejemplos de mallas estructuradas. 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 ordenador y es más fácil controlar la forma y tamaño de los elementos.

La gran desventaja que ofrecen las mallas estructuradas es su 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.

La generación automática de mallas no estructuradas es un campo relativamente reciente y que en poco tiempo ha producido grandes avances. Esta disciplina utiliza conceptos de la geometría computacional, y por consiguiente del álgebra lineal y de la teoría de poliedros entre otros campos, y sus aplicaciones se encuentran principalmente en el método de los elementos finitos. Es frecuente que, los elementos que componen las mallas no estructuradas sean 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 y en dimensión tres teselación, o también triangulación en dimensión tres. Es sabido que es posible descomponer un dominio poliédrico (poligonal en dimensión dos) en tetraedros (triángulos), aunque esta descomposición no es única y puede requerir la inserción de puntos adicionales.

1.1. Algoritmos de generación de mallas

Presentamos un breve resumen de los principales tipos de algoritmos utilizados para la generación de mallas no estructuradas. La generación de mallas no estructuradas a partir de triángulos y tetraedros es una de las técnicas más

utilizadas actualmente. La mayoría de las técnicas que se utilizan actualmente se pueden englobar en las tres principales categorías:

1. Quadtree-octree
2. Delaunay
3. Avance Frontal

Aunque hay ciertamente un grado de complejidad mayor cuando trabajamos en 3D que cuando lo hacemos en 2D, los algoritmos que aquí se exponen en gran parte son aplicables para la generación de mallas ya trabajemos en dos o tres dimensiones.

1.1.1. Quadtree-octree

La técnica Octree fue desarrollada inicialmente por el grupo de Mark S. Shephard en el Polythechnic Institute de Rensselaer, Nueva York [102, 103, 92] durante la década de los ochenta, y es una extensión del caso bidimensional conocido como Quadtree.

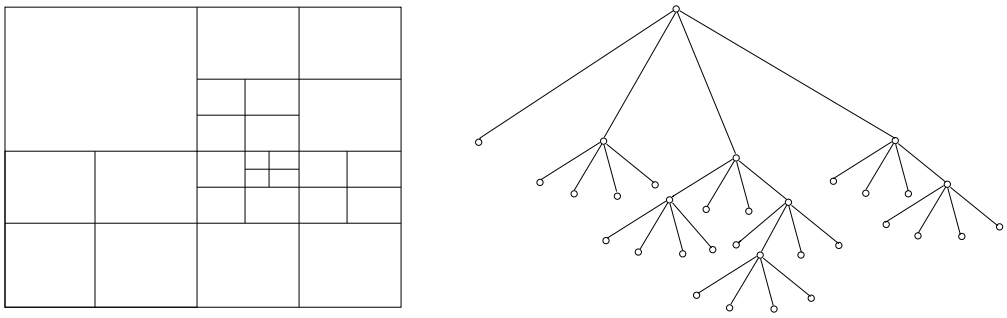


Figura 1.1: Estructura de datos tipo quadtree

En el método Octree, el concepto de quadtree (cuadrados que se dividen en cuatro, resultando una estructura de datos arbórea, como se pone de manifiesto en la figura 1.1) es reemplazado por hexaedros -o cubos- que se dividen entre cero y ocho hijos. Ahora el proceso de división es más complejo que en dimensión dos. En concreto para obtener la conformidad de la malla resultante,

todos los elementos se dividen finalmente en tetraedros.

Sea Ω un dominio acotado de $\mathbb{R}^3$ con frontera $\partial\Omega$ poligonal. Los pasos del algoritmo, de la misma naturaleza que en dimensión dos, son los siguientes:

Paso 1: Se genera un paralelepípedo inicial, que contiene todos los puntos de la frontera del dominio. El método consiste en dividir, partiendo de la malla inicial, cada caja o hexaedro-padre en algunos otros hexaedros hijos (entre cero y ocho) mediante un proceso recursivo hasta que, se obtiene la malla final. Esta malla final cumple el criterio de que cada caja final contiene como mucho un vértice de la frontera del dominio Ω. Este criterio permite controlar el tamaño de cada elemento.

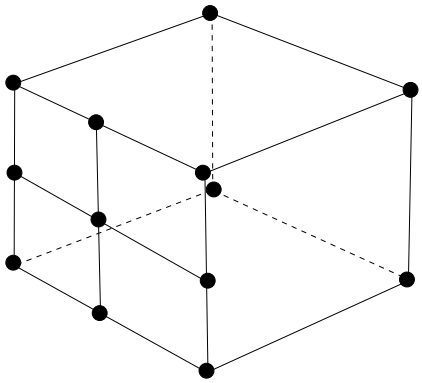
Paso 2: La partición resultante del paso anterior se puede suavizar definiendo los hexaedros hijos de tal forma que, como mucho, haya un punto intermedio en cada cara de los hexaedros.

Paso 3: Se realiza un análisis de las caras cuadrangulares teniendo en cuenta:

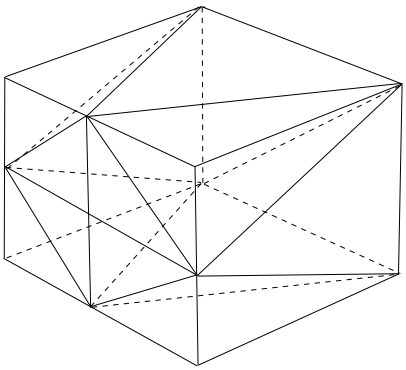
- Cara cuadrangular externa que no contenga ningún punto de la frontera de Ω es eliminada.
- Cada caja interna que no contenga ningún punto produce un hexaedro que es dividido en tetraedros, con el fin de obtener la conformidad de la malla.
- Para cada caja que contenga un *"trozo"* de frontera, se calculan los puntos intersección, y el tetraedro es dividido. Se define en este punto el contorno final de la malla.

Sin entrar en mucho detalle, merece la pena señalar que la creación de los tetraedros, partiendo de la malla de hexaedros se realiza en dos pasos:

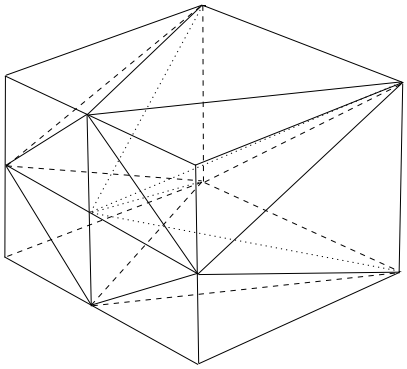
- Cada cara cuadrangular es dividida en triángulos como se muestra en la figura 1.2 (b), siguiendo ciertos *"patrones"* semejantes al caso bidimensional, de forma que la superficie formada por las caras cuadrangulares (esqueleto) se hace conforme.
- Se dividen internamente los hexaedros en tetraedros de forma congruente con la división realizada previamente en las caras (ver figura 1.2 (c)).



(a) Detalle de una malla de hexaedros



(b) División de las caras cuadrangulares en triángulos



(c) Configuración final

Figura 1.2: División de un cubo por octree

Paso 4: Se realiza una regularización de los puntos internos.

Las mallas obtenidas por esta técnica tienen, en general, una buena calidad en zonas relativamente lejanas de la frontera del dominio. Sin embargo, la calidad puede ser pobre cerca del borde. La calidad de los elementos creados depende de la lista de patrones usados y de su tratamiento posterior. Además una aproximación muy precisa de la frontera puede inducir la creación de un enorme número de elementos. La técnica Octree no necesita una malla superficial predefinida, como ocurre con Delaunay y Avance Frontal. La malla resultante cambiará cuando la orientación de los cubos en la estructura octree cambie.

Para asegurar que el tamaño de los elementos no cambie de una manera drástica, se puede limitar a uno la diferencia máxima en el nivel de división octree para cubos adyacentes. Posteriormente pueden utilizarse operaciones de suavizado con el fin de mejorar los tipos de elementos.

Finalmente, la técnica Octree puede usarse en combinación con la de Avance Frontal y con técnicas de Delaunay.

1.1.2. Delaunay

La triangulación de Delaunay se basa en el concepto propuesto por Dirichlet para la descomposición de un dominio dado en un conjunto de polígonos convexos. Dirichlet asignó a cada punto de definición del dominio, una zona o región, denominado *tesela*, que es el área del plano más cercana al punto que a cualquier otro. Estas *teselas* se conocen como polígonos de Voronoi y se definen de la siguiente forma:

Dado un conjunto de puntos en el plano $\{P_i\}$, las regiones o polígonos de Voronoi cumplen:

$$V_i = \{p : |p - p_i| < |p - p_j|,\ \forall\ j \neq i\} \tag{1.1}$$

Es decir, un polígono de Voronoi V_i es el conjunto de todos los puntos p que están más cercanos a p_i que a cualquier otro punto p_j con $i \neq j$. En $\mathbb{R}^2$, las líneas que forman una cara de un polígono de Voronoi son las mediatrices

entre los dos puntos que éstas separan. Si se unen todos los puntos que tienen común alguna cara de los polígonos de Voronoi, se obtiene una triangulación de la envoltura convexa de la unión de todos los puntos. A esta triangulación se le conoce como triangulación de Delaunay.

En el interior de cada triángulo sólo existe un punto que es un vértice de un polígono de Voronoi y en este punto sólo inciden tres caras de estos polígonos, siendo estas caras, las mediatrices de los lados de los triángulos. Esto significa que este punto es el circuncentro del triángulo asociado de la triangulación de Delaunay (criterio de Delaunay).

El criterio de Delaunay dice que ningún nodo puede estar contenido en el

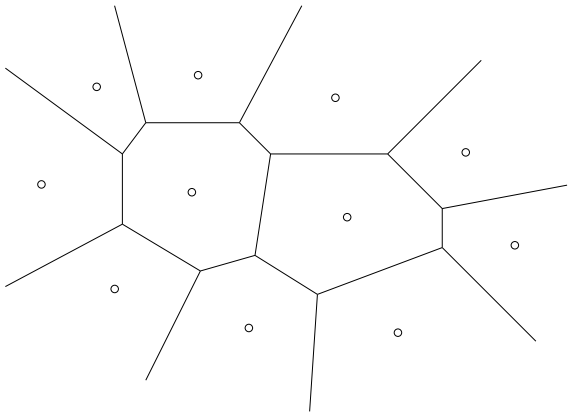


Figura 1.3: Conjunto de polígonos de Voronoi asociado al conjunto de puntos

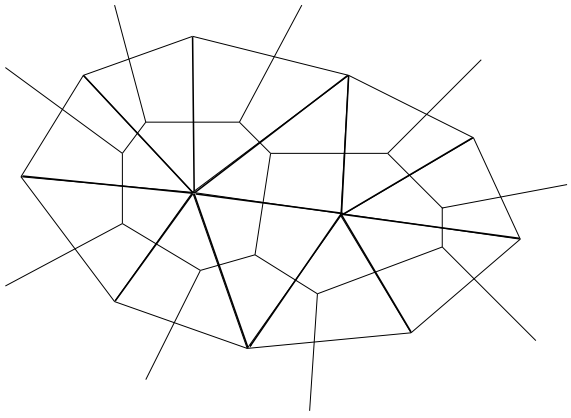


Figura 1.4: Triangulación de polígonos de Voronoi

interior de la esfera circunscrita para cualquiera de los tetraedros que conforman la malla. Una esfera circunscrita puede venir definida como la esfera que pasa por los cuatro vértices del tetraedro.

Es decir, dado un conjunto X de puntos en un espacio euclídeo de dimensión d ($d = 2$ *o* 3), la triangulación de Delaunay crea una triangulación en la que los vértices de los símplices son los puntos de X. 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. Uno de los más utilizados es el de Watson [99], cuya ventaja principal sobre otros algoritmos de tipo incremental es su simplicidad conceptual.

Para la implementación del algoritmo de Watson se ha de tener en cuenta en general que el dominio a discretizar puede cumplir que: el contorno exterior del dominio puede ser una poligonal convexa y que en su interior pueden existir agujeros (véase figura 1.5).

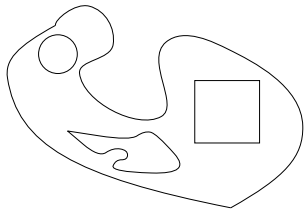


Figura 1.5: Dominio con agujeros en su interior

Supongamos un dominio cerrado y con agujeros interiores, definido por un conjunto de puntos que serán los nodos de la triangulación de Delaunay. Este conjunto de puntos lo dividimos en los siguientes subconjuntos:

1. Conjunto de puntos que definen al contorno exterior del dominio.
2. Conjunto de puntos que definen a cada uno de los contornos interiores al dominio (agujeros).
3. Conjunto de puntos que pertenecen al dominio y que no pertenecen a ninguna frontera.

El algoritmo de Watson para este dominio será:

1. Se crea una poligonal que encierre a todos los puntos. Watson utiliza un triángulo y otros autores un cuadrilátero. En general se puede usar cualquier poligonal.

2. Se aplica el algoritmo de triangulación de Delaunay al conjunto de puntos formado por los vértices de la poligonal creada en el apartado anterior y los puntos del contorno exterior.

3. De la triangulación obtenida se eliminan los triángulos exteriores al dominio.

4. Se inserta en la triangulación obtenida los puntos que definen los distintos contornos interiores (agujeros).

5. Se eliminan los triángulos interiores a los contornos interiores.

6. Una vez obtenida la triangulación asociada a los puntos que forman los distintos contornos del dominio, insertamos los puntos interiores del dominio que no forman parte de ningún contorno.

Esta técnica presenta varios inconvenientes a la hora de implementarla. Algunos de ellos son:

1. Al eliminar los triángulos en los apartados 3 y 5 los triángulos exteriores al dominio cuyos vértices pertenezcan al mismo tienen que ser marcados sin tener en cuenta aquellos triángulos con algún vértice en la poligonal cerrada en el apartado 1, ya que, aunque sean eliminados para la triangulación final, pueden intervenir en la aplicación del método de Delaunay cuando se inserten puntos posteriormente.

2. Cuando realizamos el paso 3, la triangulación asociada queda envuelta por un dominio no convexo y esto contradice la propiedad de la triangulación de Delaunay de la poligonal convexa.

3. Si los triángulos son eliminados por etapas como propone el algoritmo propuesto, podría ocurrir que algún contorno se viera modificado. Pasados los apartados 1 y 2 del algoritmo y eliminados los triángulos con algún vértice en la poligonal creada por el apartado 1, tenemos la triangulación de la figura 1.6.

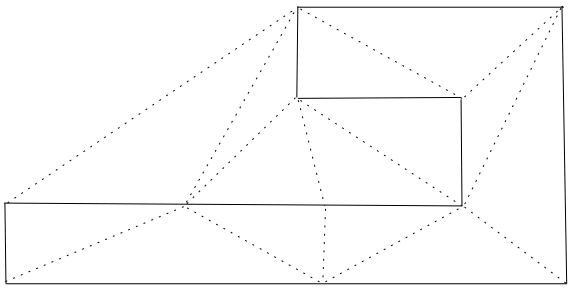


Figura 1.6: Triangulación de Delaunay después de los pasos 1 y 2

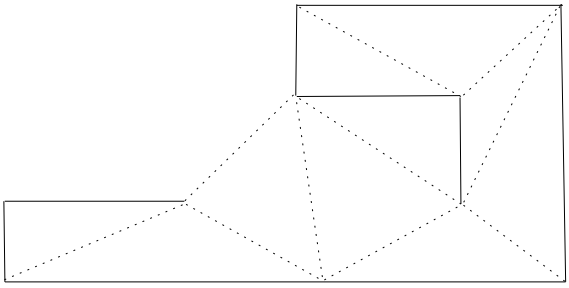


Figura 1.7: Triangulación de Delaunay realizado el paso 3

4. Por último, si eliminamos los triángulos según el apartado 3, el dominio se modificaría como se muestra en la figura 1.7.

El refinamiento de la malla se puede acometer sin necesidad de reconstruirla desde el principio, bastaría con introducir nuevos puntos, en posiciones adecuadas de la malla existente.

La triangulación de Delaunay es única siempre que en las circunferencias circunscritas a los triángulos no haya más de tres puntos del conjunto de puntos a triangular. Por ello conviene estudiar algunas degeneraciones que pudieran aparecer en la triangulación:

1. Caso en que tres puntos de un posible triángulo están alineados. Esto implica que el circuncentro de los tres puntos está en el infinito. Este fenómeno se puede solucionar eliminando un punto o moviéndolo una pequeña distancia antes de incluirlo.

2. Caso en que cuatro o más puntos están sobre una circunferencia circunscrita a un triángulo. Por las definiciones anteriores se sabe que un vértice

de un polígono de Voronoi sólo contiene tres aristas. En la figura 1.8, vemos una triangulación de Delaunay de cuatro puntos y observamos que es única.

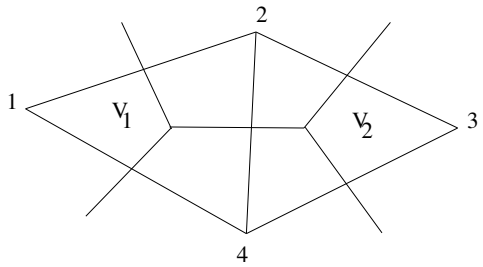


Figura 1.8: Triangulación de Delaunay

Si los cuatro puntos están sobre el círculo, como se ve en la figura 1.9 , el vértice del polígono de Voronoi interior al mismo contiene cuatro aristas, con lo cual el algoritmo nos ha dado un cuadrilátero en vez de un triángulo.

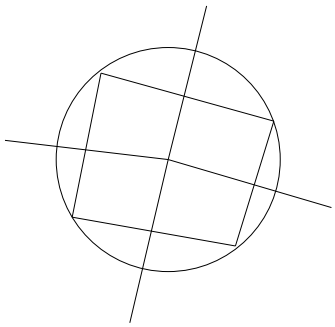


Figura 1.9: Formación de un cuadrilátero

Una solución es triangular a posteriori dicho cuadrilátero, trazando una diagonal, y por lo tanto, teniendo dos posibilidades. En este caso la triangulación de Delaunay no es única (figura 1.10).

Una de las propiedades más importantes de la triangulación de Delaunay es la de poder insertar un nuevo punto interior a una triangulación ya existente, obteniéndose una nueva triangulación de Delaunay como se ve en la figura 1.11.

Al insertar un nuevo punto habrá un conjunto de triángulos cuyo circuncírculo tendrá el punto introducido en su interior. La unión de todos estos

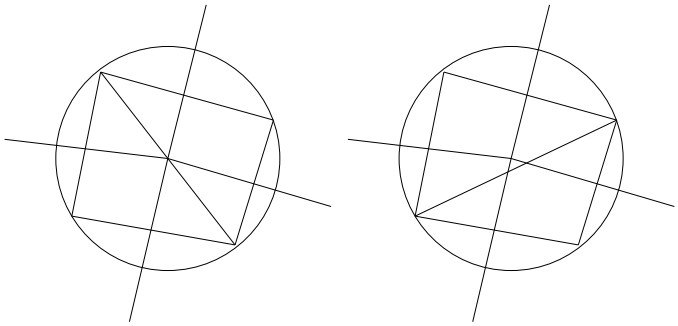


Figura 1.10: Triangulación de Delaunay, no única

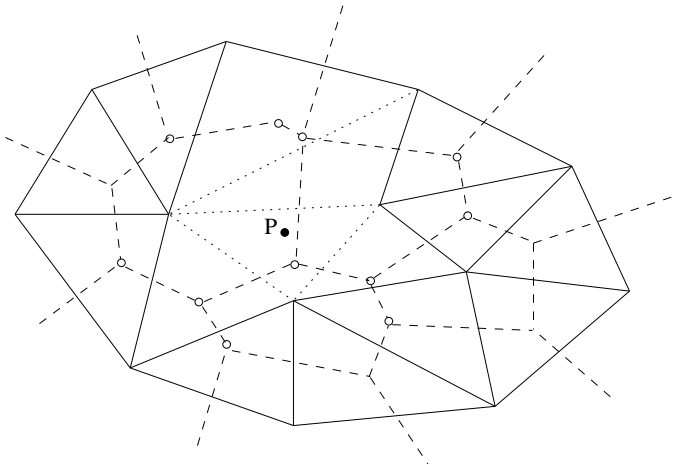


Figura 1.11: Inserción de un nuevo punto

triángulos forman un polígono denominado polígono de inserción, como se ve en la figura 1.12.

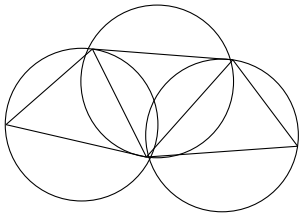


Figura 1.12: Formación de un polígono de inserción

La nueva triangulación de Delaunay estará formada por todos los triángulos de la anterior triangulación, exteriores al polígono de inserción, más los triángulos formados al unir dicho punto con los vértices del polígono de inserción.

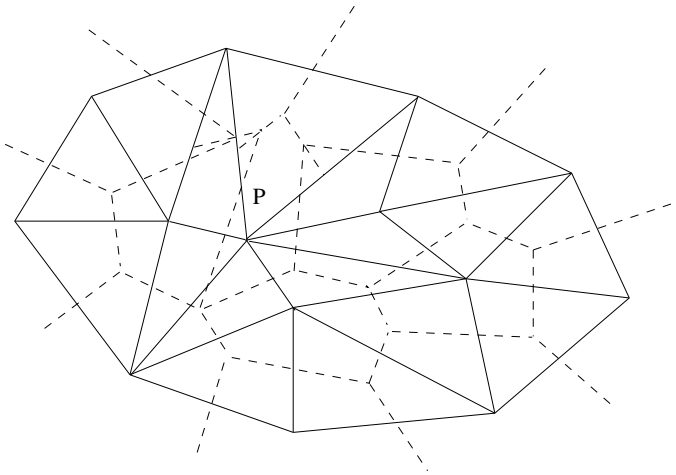


Figura 1.13: Nueva triangulación de Delaunay

La triangulación de Delaunay reúne también algunas características que la hacen adecuada para la aplicación del Método de Elementos Finitos (M.E.F.). Además de la capacidad de refinamiento, la triangulación de Delaunay posee propiedades geométricas óptimas: 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 [24]. 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*), o 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*).

También en dimensión tres hay algoritmos (ver por ejemplo [31, 100]) 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, ó bien utilizar la triangulación de Delaunay y el refinamiento mediante la técnica de insertar los nuevos vértices en los puntos medios de las aristas [84].

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 esfera de dimensión d. 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*. Algunas técnicas que resuelven este problema se han propuesto en [20].

1.1.3. Avance frontal

El método de avance frontal recibe su nombre del hecho de que en cada etapa del proceso existe un frente formado por todos los posibles lados sobre los cuales se puede generar un nuevo elemento. Los generadores de mallas que utilizan el método de avance frontal han conseguido gran aceptación gracias a su versatilidad y velocidad [11, 41, 49, 51, 60, 65]. La idea básica del método consiste en marchar hacia el espacio que aún no ha sido mallado, introduciendo un elemento cada vez. La región que separa la parte del espacio que ya ha sido mallada de aquella que aún no ha sido mallada se denomina *frente*. La complejidad del algoritmo de avance frontal es de orden $\mathcal{O}(N \log(N))$, donde N es el número de elementos. De forma repetitiva el proceso consta de una etapa de generación de un elemento y de otra etapa de actualización del frente.

El primer frente o línea de avance está formada por los puntos generados en el contorno exterior del dominio. Para la formación de un elemento son conocidas dos técnicas distintas:

1. Formación de un elemento dada una discretización del interior del dominio. Técnica propuesta por S.H. Lo [50].

   Dados dos nodos A y B consecutivos en la línea de avance, es elegido un nodo (C_1 o C_2) de la discretización interior del dominio o de la línea de avance con el siguiente criterio:

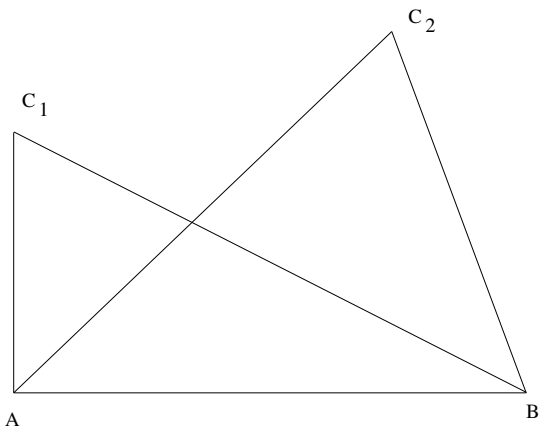


Figura 1.14: Elección de un nodo cuando el frente avanza

a) Para el nodo C_1 se define:

$$\alpha_1 = \frac{\Delta ABC_1}{AB^2 + BC_1^2 + CA_1^2} \tag{1.2}$$

$$\beta_1 = \frac{\Delta C_1BC_2}{C_1B^2 + BC_2^2 + C_1C_2^2} \tag{1.3}$$

$$\gamma_1 = \frac{\Delta AC_1C_2}{AC_1^2 + C_1C_2^2 + C_2A^2} \tag{1.4}$$

$$\lambda_1 = max(\beta_1, \gamma_1) \tag{1.5}$$

b) Para el nodo C_2 se define:

$$\alpha_2 = \frac{\Delta ABC_2}{AB^2 + BC_2^2 + C_2A^2} \tag{1.6}$$

$$\beta_2 = \frac{\Delta C_2BC_1}{C_2B^2 + BC_1^2 + C_1C_2^2} = -\beta_1 \tag{1.7}$$

$$\gamma_2 = \frac{\Delta AC_2C_1}{AC_2^2 + C_2C_1^2 + C_1A^2} = -\partial_1 \tag{1.8}$$

$$\lambda_2 = max(\beta_2, \gamma_2) \tag{1.9}$$

c) Se elige C_1 si:

$$\alpha_1\lambda_1 \leq \alpha_2\lambda_2 \tag{1.10}$$

En caso contrario sería seleccionado el nodo C_2.

2. Otra técnica básica que permite controlar el tamaño de los elementos y su forma es la siguiente. Dados dos nodos A y B consecutivos en la línea de avance, es elegido un nodo C de la siguiente forma:

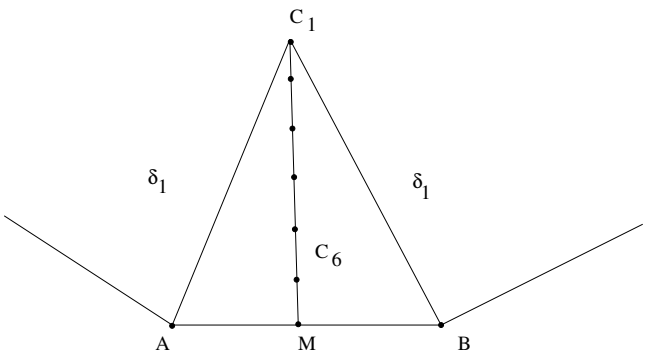


Figura 1.15: Determinación de puntos equiespaciados

a) Elección de un tamaño de elemento δ.

b) Determinar un punto C_1 a una distancia δ_1 de A y B con el siguiente criterio:

$$\delta_1 = \delta \text{ si } 0{,}55AB < \delta < 2AB \tag{1.11}$$

$$\delta_1 = 0{,}55AB \text{ si } 0{,}55AB > \delta \tag{1.12}$$

$$\delta_1 = 2AB \text{ si } \delta > 2AB \tag{1.13}$$

c) Determinación de los puntos C_2, C_3, C_4, C_5 y C_6 que estarán equiespaciados en el segmento que unirá los puntos C_1 y M.

d) Determinación de los nodos pertenecientes a la línea de avance que estén situados dentro de un círculo de radio $r = 3AB$ y centro C_1. De estos nodos se escogen los que estén situados en el semiplano delimitado por la recta que une los puntos A y B y que contienen al punto C_1. Estos nodos se ordenan según la distancia al punto C_1 y se adjunta a una lista de tal forma que el primero será el más cercano aplicándose a todos ellos una penalización de valor δ_1.

e) Determinación del punto de conexión C. Este será el primer nodo de la lista que satisfaga las dos condiciones siguientes:

1) El interior del triángulo ABC no contiene a ninguno de los restantes nodos de la línea de avance.

2) El segmento CM no corta ninguna de las caras existentes en ese momento en la línea de avance.

1.2. El Método de los Elementos Finitos

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 resulta imposible. El interés práctico que tiene conocer soluciones aproximadas de estas ecuaciones, y la rapidez en los cálculos que actualmente proporcionan los ordenadores, han hecho que los métodos numéricos cobren gran importancia. En concreto, el método de los elementos finitos es uno de los más usados para la simulación numérica de problemas físicos variados, formulados en términos de ecuaciones diferenciales (problemas de cálculo de temperaturas, desplazamientos, presión, velocidades,

campos magnéticos, etc.).

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 subdominios de geometría simple, los elementos. Para una discretización geométrica considerada, podemos utilizar distintos elementos finitos que geométricamente sean iguales. En el caso de elementos finitos geométricamente distintos, debe prestarse especial atención a verificar la conformidad del mallado, o aceptar la no conformidad en la solución. El conjunto de estos subdominios se denomina *malla* del dominio. Esta fase del preproceso es muy importante en la medida en que la generación de un mallado para un dominio de geometría compleja no es una operación simple y además la calidad de la solución numérica depende del mallado considerado. Así, mallados con características no aceptables repercuten en la no convergencia del método de los elementos finitos, o bien hacen disminuir la velocidad de convergencia. También es cierto, que la calidad de la solución numérica está limitada por la densidad y la calidad de la discretización. Para problemas con singularidades, dependiendo del tipo de singularidad, existen además elementos finitos especiales que mantienen la misma regularidad caso de que el problema fuera regular. Sin el uso de métodos adaptativos, la malla caso de ser irregular, debe atender cualitativamente al comportamiento de estimadores de error locales para reducir el efecto de polución en los elementos cercanos a los que contienen nodos o fronteras en las singularidades.

La malla debe acercarse lo más posible a algunos requisitos generales, tales como:

1. Adaptarse lo mejor posible a la forma del objeto. En este sentido, las mallas deben adaptarse a las singularidades, haciendo coincidir nodos y contornos de elementos con las singularidades y cargas puntuales, importantes en resolución p-adaptativa, h-adaptativa o h-p-adaptativa.

2. Presentar mayor densidad de elementos en las zonas donde la solución varíe más rápidamente.

3. La transición de una zona donde los elementos sean *grandes* a otra donde

sean *pequeños* debe ser gradual. Por tanto, las mallas irregulares deben evitar bruscas variaciones de tamaño entre un elemento y sus adyacentes, siendo de interés el diseño de criterios flexibles de definición de densidades de mallados en el dominio.

4. 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.

1.2.1. Formulación del problema

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

$$''Hallar\ u \in V\ tal\ que : a(u,v) = f(v)\ \forall\, v \in V'' \tag{1.14}$$

donde V es un espacio de Hilbert, $a : V \times V \rightarrow I\!R$ es una forma bilineal continua y elíptica, y $f : V \rightarrow I\!R$ es una forma lineal y continua. El teorema de Lax-Milgram asegura entonces la existencia y unicidad de la solución. Sin embargo, en la mayor parte de los casos prácticos no se puede encontrar la solución exacta del problema anterior; se buscan entonces soluciones aproximadas.

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

$$''Hallar\ u_h \in V_h\ tal\ que : a(u_h,v_h) = f(v_h)\ \forall\, v_h \in V_h\,'' \tag{1.15}$$

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

Ejemplos típicos de espacios V que aparecen en las aplicaciones son $H^1(\Omega)$, $H_0^1(\Omega)$, $H^2(\Omega)$, $H_0^2(\Omega)$, etc. Los problemas de contorno asociados a ecuaciones

en derivadas parciales elípticas de segundo a cuarto orden lineales son ejemplos que se pueden resolver siguiendo el esquema abstracto anterior.

El método de elementos finitos, en su forma más sencilla, es un método específico para construir subespacios de dimensión finita de V, con el objetivo de hallar soluciones aproximadas de problemas de contorno. El método comprende la división del dominio en el cual se encuentra definido nuestro problema en subdominios, denominados elementos finitos, y el uso de formulaciones débiles o variacionales para la obtención de una solución aproximada sobre la colección de elementos finitos. La gran generalidad y variedad de las ideas que subyacen en el método permite aplicar éste, con pleno éxito, en un amplio espectro de problemas en casi todas las áreas de la ingeniería y de la física matemática. Por otra parte, la aplicabilidad del método no se limita a los problemas mencionados sino que se extiende a problemas parabólicos, hiperbólicos, no lineales, etc.

El punto de partida, y a su vez el aspecto más característico del método de elementos finitos, es la subdivisión del dominio Ω , en el que está planteado el problema a resolver, en subdominios $\mathcal{K}$ mediante, por ejemplo, una *triangulación* τ del mismo, de modo que se cumplan las siguientes propiedades:

1. $\bar{\Omega} = \bigcup_{\mathcal{K}\in\tau} \mathcal{K}$
2. $\forall\, \mathcal{K} \in \tau$, $\mathcal{K}$ es cerrado y su interior $\overset{\circ}{\mathcal{K}}$ es no vacío
3. Para cada par $\mathcal{K}_1,\ \mathcal{K}_2 \in \tau,\ \overset{\circ}{\mathcal{K}}_1 \bigcap \overset{\circ}{\mathcal{K}}_2 = \emptyset$

Finalmente, un elemento finito, siguiendo el formalismo introducido por Ciarlet [14], está caracterizado por una terna $(\mathcal{K}, \mathcal{P}, \Sigma)$ donde

i) $\mathcal{K}$ es un conjunto cerrado de $\mathbb{R}^d$ de interior no vacío y de frontera $\partial\mathcal{K}$ lipschitziana, donde d es la dimensión de dominio Ω.

ii) $\mathcal{P}$ es un espacio de funciones reales definidas sobre $\mathcal{K}$.

iii) Σ es un conjunto finito de formas lineales independientes definidas sobre $\mathcal{P}$.

1.2.2. El problema del refinamiento local

Además de la generación de la malla (inicial), se puede considerar el problema del refinamiento local. Un refinamiento de mallado excesivamente fuerte, aún a nivel local, puede producir inestabilidad numérica en la resolución, debiéndose atender a trabajar en doble precisión y a la resolución de sistemas mal condicionados. Como ya se ha dicho, en muchos problemas de ingeniería que simulan situaciones reales es difícil prever cuál sería la malla apropiada. Se pueden realizar varios intentos con el fin de obtener la solución más aproximada si se tiene un límite máximo en cuanto al tamaño de los elementos utilizados o sin utilizar un excesivo número de nodos.

En un proceso adaptativo hay dos fases: análisis del error cometido con la malla anterior y refinamiento de la malla si procede. Los análisis del error a posteriori nos permiten calcular para cada elemento una cota del error total cometido -en el caso de estimadores de error-, o bien -en el caso de indicadores de error- nos proporcionan un valor por elemento relativo a otros elementos de la malla. En los métodos adaptativos, la utilización de distintos indicadores de error (en problemas elípticos sobre todo) permite plantear estrategias de resolución adaptativas de refinamientos locales para que cada elemento de la malla tenga un error relativo muy similar, de tal forma que el error global sea menor que el deseado.

Muchos de los primeros trabajos de análisis del error a posteriori fueron debidos a Babuška y Rheinboldt [3, 4]. Otros trabajos importantes en este aspecto son los debidos a Kelly, Gago *et al.* [26]. En esta referencia se ofrecen algunas estrategias sobre el uso de los estimadores de error a la hora de refinar una malla. Zienkiewicz y Zhu señalaban después [105] una estimación del error simplificada que no requería el cálculo de saltos de flujo en las caras. Sin embargo el campo del análisis de errores esta aún abierto en muchos problemas en los que no se dispone de estimadores de error, y, además, donde la eficacia de los estimadores propuestos es discutida.

Una vez que se ha obtenido una estimación o indicación del error en cada elemento, se pueden seguir varias estrategias para adaptar la malla. Las dos posibilidades básicas son el p-refinamiento (o refinamiento en p) y el h-refinamiento (o refinamiento en h). El p-refinamiento consiste en un aumento

en el grado de la aproximación polinomial dentro de cada elemento finito. En general, el refinamiento con aumento de p no sobrepasa $p = 3, 4$ y el coste computacional es limitado. El refinamiento en h significa simplemente una reducción en el tamaño de subdivisión de la malla. El refinamiento en p, especialmente cuando se combina con una formulación jerárquica y con una malla suficientemente adaptada, tiene algunas ventajas: es más eficiente, converge más rápidamente. En la versión del p-refinamiento, el cálculo de la matriz local es paralelizable y permite el uso de fórmulas de cuadratura con mayor número de puntos de interés en ciertos elementos curvilíneos. Por otro lado, genera matrices menos dispersas con el consiguiente aumento del coste computacional. También es cierto que incorporar un refinamiento en p a un código ya existente generalmente implica una re-estructuración del mismo, cuando no una total re-escritura.

El refinamiento en h es más sencillo y se entiende de forma intuitiva. Es fácil de programar en un código y ha sido ampliamente aceptado. Dentro del refinamiento en cuanto al tamaño de los elementos, caben dos posibilidades: regeneración de la malla o subdivisión de los elementos ya existentes. Podemos llamar a la primera posibilidad *remalladores* y a la segunda *mallados encajados no estructurados*.

La alternativa de los remalladores implica regenerar completamente la malla, o bien sólo aquellas regiones con alto error. La ventaja de regenerar toda la malla es que las áreas en las que el error está por debajo del permitido se pueden hacer más groseras: en ellas se pierden nodos, es decir se da una especie de desrefinamiento (véase, por ejemplo, [106]). De esta forma, es posible que el número de nodos en la malla refinada sea menor que en la malla anterior. En este sentido, el refinamiento en h es más flexible en mallados no encajados que en mallados encajados.

Entre los remalladores que utilizan elementos triangulares está el avance frontal (por ejemplo [16]), aunque han aparecido trabajos que dejan abierta la posibilidad de utilizar el avance frontal incluso con elementos rectangulares [104] si bien con un menor grado de flexibilidad a la hora de colocar los nodos donde sea preciso, y también de convertir mallados triangulares en rectangulares automáticamente [42]. Otro mallador de este tipo es el conocido

método de triangulación de Delaunay.

Con la subdivisión de elementos, cada elemento que excede el error admisible prefijado es subdividido en elementos más pequeños de alguna forma. En este sentido se pueden citar, por ejemplo, los trabajos de Bank y Sherman [5, 6] y de Rivara [79, 80, 81]. Mediante el refinamiento en mallados encajados no estructurados, se crea un nuevo nivel de subdivisión de la malla cada vez que se refina, pero no se eliminan nodos. Es decir, el número de nodos va en aumento con el número de refinamientos realizados. Esto es particularmente importante en problemas en los que se requieren zonas de refinamiento móviles, como pueden ser problemas dependientes del tiempo: quasi-evolutivos o evolutivos. Muchos de los nodos que se introdujeron en la malla en instantes de tiempo pasados, pueden resultar en el momento actual innecesarios e incluso perjudiciales al aumentar de forma superflua el número de ecuaciones que hay que resolver.

Por otro lado, entre las ventajas de los mallados encajados no estructurados, la principal es que es mucho más fácil utilizar métodos multimalla para resolver el sistema de ecuaciones asociado al método de elementos finitos, con las ventajas de coste computacional que lleva consigo el método multimalla, sobre todo en problemas con gran número de grados de libertad (véase [33, 34]). Además el proceso de refinamiento es más rápido, y, en ciertos problemas, no se necesitan recalcular las matrices elementales de aquellos elementos que no se refinan; esto último es debido a la estructura de datos asociada al proceso de refinamiento en los diferentes niveles de malla. Otra interesante ventaja de los mallados encajados es que se consigue un cierto control sobre la geometría de las mallas obtenidas, pues ésta depende de la geometría de la malla inicial. Se evita de esta forma la degeneración del mallado, es decir, la aparición de ángulos cercanos a cero, lo que conlleva algunas dificultades numéricas.

En algún caso particular de refinamiento en h en mallados encajados, se ha desarrollado un algoritmo de desrefinamiento [22, 66, 81], capaz de eliminar nodos del mallado. Recientemente, estos algoritmos se han generalizado también a tres dimensiones [63].

1.3. Representación gráfica en 2D y 3D

Otras muchas aplicaciones de la generación de mallas se pueden reunir bajo el nombre de representación gráfica, aunque también está muy extendido el nombre de modelización gráfica [38, 90]. Numerosas disciplinas relacionadas con la representación gráfica de objetos 2D y 3D han surgido en las últimas décadas. Cada vez más se demandan eficientes y rápidos sistemas de software para la visualización, cálculo, simulación y modelado de objetos en 2D y 3D. En todos ellos, el problema en común es la representación gráfica por ordenador de los objetos en 2D y 3D que tenga ciertas características como:

1. Reducido tiempo de cómputo y reducido almacenamiento, que está directamente relacionado con los métodos de representación, algoritmos y estructuras de datos utilizados en la implementeación.

2. Permitir al usuario una interacción con las representaciones gráficas que sea simple, consistente y robusta.

3. Que la visualización de la información gráfica sea rápida y realista.

Aunque la tecnología tanto de hardware como software avanza cada vez más, todavía es notable que no existe un entorno potente altamente eficaz que tenga las características anteriores de forma conveniente y a costo aceptable. Asimismo, el interés científico en las materias relacionadas con la representación gráfica, como por ejemplo, geometría computacional, gráficos y visión por ordenador, análisis numérico, CAD/CAM, etc. sigue dedicando gran esfuerzo en este sentido y son muchos los trabajos publicados en las últimas décadas.

En la figura 1.16 se muestra un ejemplo de aplicación de la generación inicial con malla Delaunay y refinamiento de zonas de la malla [94, 95]. El dominio para este problema es la isla de Gran Canaria, e ilustra cómo la combinación de un malla inicial Delaunay y diferentes aplicaciones de refinamiento permiten realizar una representación gráfica en 2D resaltando zonas de especial interés, en este caso las zonas de la isla con mayor irregularidad en su relieve.

En la figura 1.17 se presenta otra aplicación de la generación de mallas en combinación con el proceso de refinamiento [63] a la aproximación de superficies convexas. Muestra cómo a partir de un dominio inicial de entrada

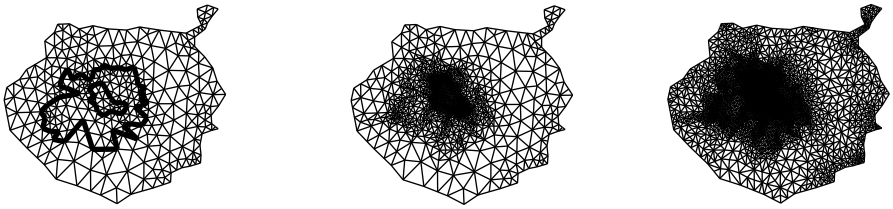
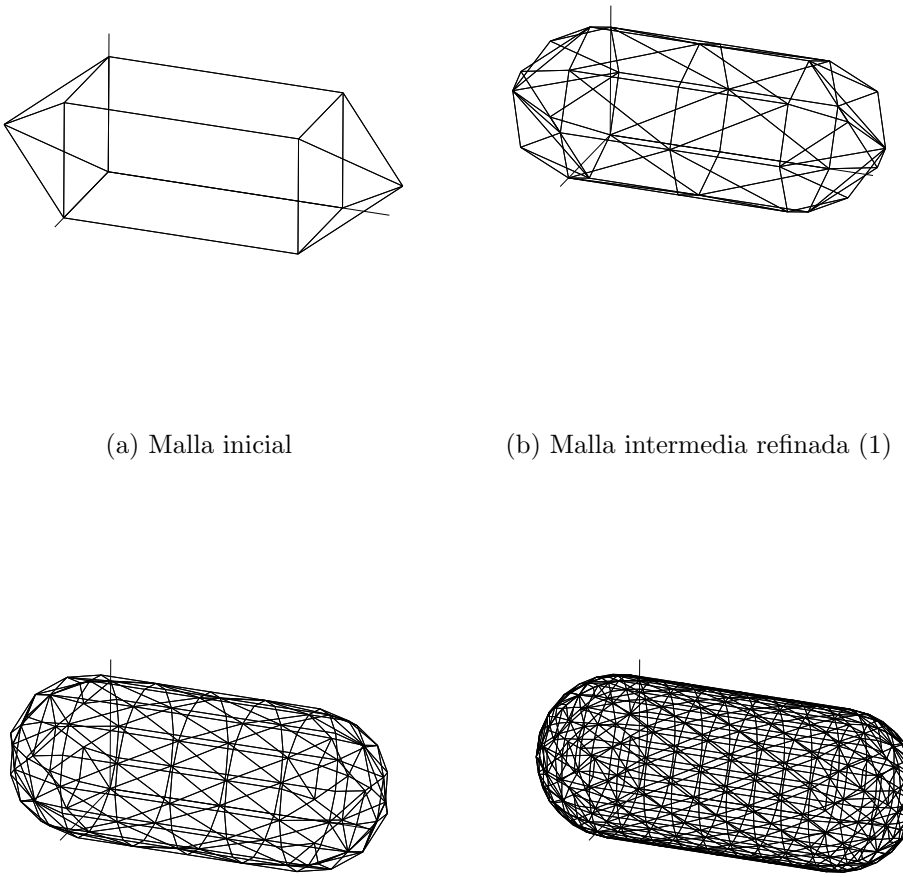


Figura 1.16: Triangulación de la superficie de Gran Canaria

(figura 1.17 (a)) se realizan diferentes procesos de refinamiento hasta obtener una malla final (figura 1.17 (d)). El proceso de refinamiento, aplicado a la superficie del objeto de la cual se conoce la ecuación paramétrica que la define, permite aproximar cada vez más a la superficie deseada, para lo cual es decisivo la reducción del tamaño de elemento.

El mismo ejemplo ilustrado en la figura 1.17 permite enunciar otra aplicación dirigida al campo de la visualización gráfica en concreto, visualización con niveles de detalle LOD (Levels Of Detail). Según la idea de los niveles de detalle de los elementos gráficos, es posible interpretar un objeto gráfico variando la resolución o detalle de su representación. Así, las cuatro instancias gráficas de la figura 1.17 pueden interpretarse como distintos niveles de detalle que permiten visualizar el mismo objeto gráfico. Las distintas instancias se usarán en función de diferentes requerimientos de tipo tecnológico, o si no se dispone de la tecnología potente que permita una visualización rápida y manejable del objeto, o bien, en ámbitos de transmisión gráfica por red, donde se hace impracticable aceptar grandes niveles de detalle. En el capítulo de aplicaciones se desarrollará más ampliamente este tema.

Otro ejemplo significativo que será tratado en detalle en esta tesis en el capítulo de aplicaciones, la generación de mallas es aplicada para realizar modelos digitales del terreno, DEM (Digital Elevation Models) mediante elementos triangulares. Un DEM permite modelar una extensión geográfica con diferente nivel de detalle. En la figura 1.18 se presenta el DEM de una zona de la costa de Agaete, Gran Canaria, para el cual se utiliza una generación de malla Delaunay con una disposición de puntos uniformes y equiespaciados cada 18 metros.



(a) Malla inicial

(b) Malla intermedia refinada (1)

(c) Malla intermedia refinada (2)

(d) Resultado de la aproximación

Figura 1.17: Aproximación de una superficie curva en 3D. Figura tomada de [63]

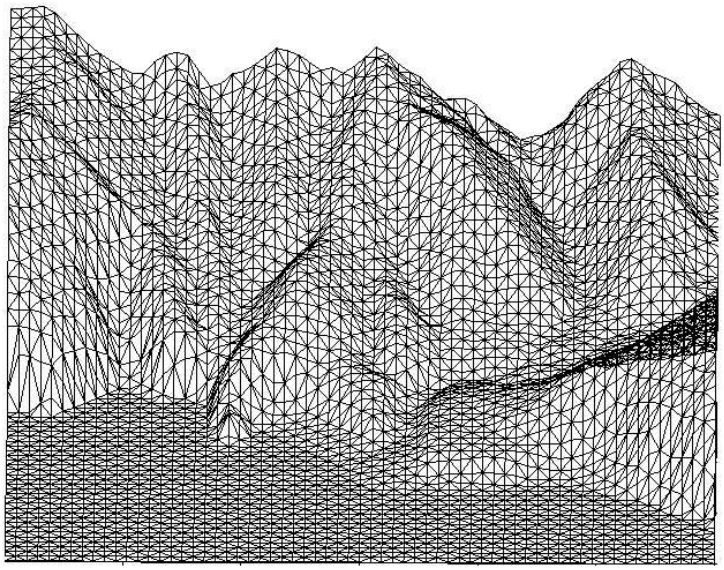


Figura 1.18: Triangulación Delaunay del terreno

1.4. Estrategias de refinamiento

En general, un código de refinamiento adaptativo sigue el esquema que se presenta en la figura 1.19. Hay que señalar que, en el marco de los mallados encajados, refinamiento adaptativo implica la inclusión de nuevos nodos en el mallado.

Para lograr la conformidad de la malla, se podría realizar lo que hacen Rheinboldt y Metsztenyi en su trabajo sobre mallas no conformes [78]. Ellos, para mantener la continuidad de la solución sobre los nodos no conformes, imponen la condición de que la solución en estos nodos sea igual a la solución interpolada de otros nodos no conformes. Parece, de todas formas, que la aparición de estos nodos no-conformes complicaría el cálculo de las matrices de rigidez elementales al tener que distinguir cada vez si un nodo es o no conforme.

El algoritmo 4-T de Rivara consiste en dividir un triángulo en cuatro del siguiente modo: en primer lugar se divide el triángulo inicial mediante bisección por el lado mayor (bisección generalizada) y después se une el nuevo vértice mediante segmentos paralelos a los otros dos lados por el punto medio para

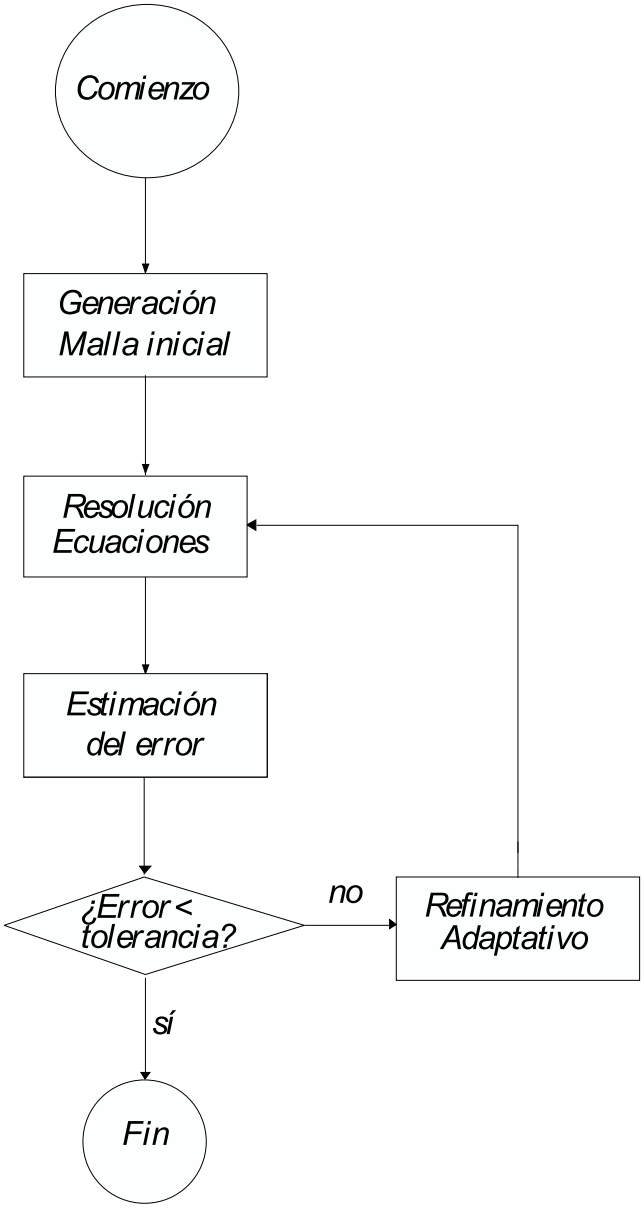


Figura 1.19: Diagrama de flujo de un refinamiento adaptativo

formar cuatro triángulos.

Los algoritmos de refinamiento y desrefinamiento del tipo 4-T de Rivara en dimensión dos fueron incorporados al código de Neptuno [21] donde se incluyen algunas estrategias de refinamiento. Por ejemplo se pueden combinar los algoritmos 2-T y 4-T como hacen Ferragut *et al.* [23]. Sea τ una triangulación y N el número de elementos de τ. Supongamos definido un cierto indicador de error por elemento, es decir, para cada triángulo $t \in \tau$ se conoce el valor de ese indicador de error: η_t. Con objeto de comparar el indicador de error local con alguna medida del error global, consideremos el número η_{opt} definido como:

$$\eta^2{}_{opt} = \frac{1}{N} \sum_{t \in \tau} \eta_t{}^2 \tag{1.16}$$

Entonces se puede seguir la estrategia siguiente:

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

b) en caso contrario, si $\eta_t \geq \beta \eta_{opt}$ con $\beta = 0{,}5$, se divide en dos.

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

Otra estrategia de refinamiento incorporada al código de Neptuno es la siguiente:

Sea

$$\eta_{max} = \max_{t \in \tau}(\eta_t) \tag{1.17}$$

Si $\eta_t \geq \gamma \eta_{max}$, siendo $\gamma \in [0, 1]$ un parámetro elegido por el usuario antes de ejecutar el programa, refinamos en cuatro triángulos. Es decir, esta estrategia nos permite refinar los triángulos cuyo indicador de error esté por encima de un cierto tanto por ciento del indicador máximo. Lógicamente, cómo se elija un indicador u otro es de capital importancia para el éxito de una estrategia adaptativa, así como la elección del parámetro γ.

Por último, se puede aplicar también la siguiente estrategia, que no es más que una modificación de la anterior. Con la misma definición de η_{max} que antes,

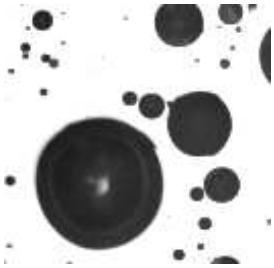
proporcionamos al programa un γ_{min} y un γ_{max}, ambos entre 0 y 1. Además fijamos el número de nodos aproximado que, como máximo, estamos dispuestos a introducir al refinar y sea éste NN_{opt}. Sea, además, NN el número de nodos en un determinado momento del mallado. Entonces el programa calcula el parámetro de refinamiento γ perteneciente al intervalo $[\gamma_{min}, \gamma_{max}] \subset [0, 1]$ más cercano al cociente $\frac{NN}{NN_{opt}}$. Se consigue de esta forma una mayor libertad en cuanto a los sucesivos refinamientos. Esta variación del parámetro depende del número de nodos NN_{opt} que es estimado *a priori* por el usuario. Esto puede ser útil sobre todo en procesos en los que el número de nodos puede aumentar o disminuir demasiado, es decir, procesos que hayan incorporado el desrefinamiento [66].

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

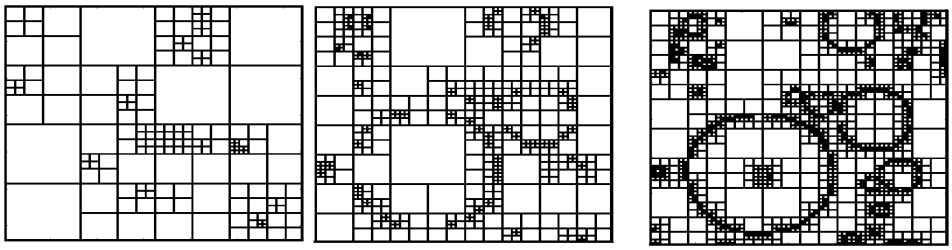
Por otra parte, el problema de refinamiento local también puede extenderse a mallas estructuradas y a otro ámbito que no sea exclusivamente el de los elementos finitos. Por ejemplo en la figura 1.20 se presenta un ejemplo de h-refinamiento tipo remallado, sobre una malla estructurada quadtree. El ejemplo fue realizado con el software *Matlab* v.5.0 con la rutina *qtdecomp* y centra su atención en la detección de bordes de una imagen digital de entrada. Se presentan tres mallas distintas correspondiendo cada una a un valor de 0.83, 0.78 y 0.28 que mide grado de detección deseado (entre 0 y 1).

1.5. Estructuras de datos geométricas

El diseño de estructuras de datos puede considerarse como un área de consideración especial en las ciencias de la computación y matemática aplicada. La organización de la información de cara a la implementación de algoritmos por ordenador exige un esfuerzo necesario de diseño que incide claramente en la eficiencia del algoritmo. Teniendo en cuenta que la información a represen-



(a) Imagen digital
de entrada



(b) Diferentes casos de remallado

Figura 1.20: Refinamiento en la detección de bordes

tar (tipo de datos) por las estructuras de datos puede ser de muy diversas características, como por ejemplo, texto, geométrica, gráfica, etc. también se exige un esfuerzo de análisis asociado al tipo de datos. Es decir, es necesario un estudio de las características específicas del tipo de datos, que satisfaga las necesidades del algoritmo que trata con ellas.

El problema de organizar y estructurar los datos en forma de estructuas de datos, fue resuelto en la era inicial de la computación de forma simple. La simplicidad era respuesta necesaria ya que los dispositivos de entrada de datos al ordenador eran sencillos. Además, los primeros lenguajes de programación ofrecían posibilidades limitadas en la estructuración y manejo de los datos. Sin embargo, con el avance de la tecnología informática y el advenimiento de modernos y sofisticados lenguajes de programación, esta respuesta no es inmediata y ofrece múltiples consideraciones y posibilidades. Puede hablarse de que se ha pasado a una sofisticación de las estructuras de datos motivada por la necesidad de dar solución a problemas de creciente complejidad.

Los problemas de estructuración de datos que hacen referencia a objetos geométricos varían y pueden clasificarse en:

1. Estáticos. En este caso, todos los objetos geométricos en el dominio del problema se proporcionan como parte de la entrada.

2. En-línea (On-line). Donde se permite añadir nuevos objetos geométricos pero no permiten ser eliminados hasta que el problema finalice.

3. Dinámicos. Este es el caso más general e implica que se permite añadir nuevos objetos así como eliminar otros que ya no hacen falta en la solución del problema.

A continuación se enumera una serie de objetivos deseables de las estructuras de datos que tratan con objetos geométricos:

1. Capturar la información estructural (por ejemplo de la malla).

2. Permitir un procesamiento eficiente de las consultas de información.

3. Permitir actualizaciones de la información lo más eficiente.

4. Optimizar el espacio de almacenamiento.

5. Almacenamiento eficiente en el sentido de minimizar el número de accesos de entrada/salida, cuando el volumen de datos es grande.

Concretamente, en el campo de la generación y refinamiento/desrefinamiento de mallas, la necesidad de una buena estructura de datos es crucial debido, entre otras, a las siguientes causas:

1. Entidades de origen espacial (geométrico y topológico).

2. Complejidad creciente en la dimensión del problema (2D, 3D ...).

3. Volumen de datos considerable, que oscila entre miles y millones de elementos.

4. Creación, eliminación y actualización de forma dinámica de los elementos.

Todo esto ha hecho surgir áreas específicas de diseño de estructuras de datos. En el campo que nos ocupa se denominan estructuras de datos geométricas y estructuras de datos espaciales.

Estructuras de datos clásicas como vectores, listas, árboles o grafos, no son por sí mismas muy adecuadas para la representación de objetos geométricos, ya que fundamentalmente están concebidas para problemas en dimensión 1 o bien no son capaces de representar las características espaciales de los objetos geométricos ni del algoritmo que las utiliza. Por ejemplo, en elementos finitos habitualmente es necesario mantener un orden determinado (sentido horario o antihorario) de las aristas dentro de un triángulo, o de las caras dentro de un tetraedro. Otras veces, por necesidad de los algoritmos de refinamiento o desrefinamiento, se necesita tener fácil acceso a los elementos vecinos a uno dado. Este tipo de necesidades se complican al aumentar la dimensión del problema, como se ha apuntado anteriormente. De esta forma, es preciso que la combinación adecuada de estructuras de datos o incluso el diseño de otras nuevas se adapten a los requerimientos del problema en cuestión.

Otro aspecto a tener en cuenta es que, por regla general, el diseño de estructuras de datos viene estrechamente relacionado con las características

del algoritmo que las use. Así, un algoritmo que por ejemplo implemente la generación y/o refinamiento de mallas para elementos finitos tipo Delaunay tendrá unas particularidades en el contexto de la información geométrica a manejar que no coincide con otros algoritmos como el Avance Frontal u Octree. Ello implica que las estructuras de datos de dichos algoritmos no podrán ser las mismas a cierto nivel de detalle.

Por ello se requiere un cuidadoso diseño de las estructuras de datos, que además sean lo más genéricas posibles, pudiéndose así extender su uso a diversos problemas afines. En el capítulo 3 se hace un estudio y recorrido por las estructuras de datos que en la literatura se usan en el contexto de la generación de mallas y refinamiento/desrefinamiento.

1.6. Aportación y objetivos de esta tesis

La aportación fundamental de esta tesis es el desarrollo y estudio de una clase de estructuras de datos geométricas basadas en la teoría de grafos. Dichas estructuras de datos surgen de forma natural de un concepto geométrico, esqueleto de una malla, que también es estudiado en esta tesis. De las nuevas estructuras de datos introducidas surgen nuevas versiones de los algoritmos 2D-SBR y 3D-SBR, que siguen siendo de complejidad lineal y permiten reducir el costo de espacio y asimismo admiten una más clara y completa descripción.

Objetivos

A continuación se presentan los cuatro objetivos fundamentales que se persiguen en esta tesis:

1. Desarrollo de unas estructuras de datos tipo grafo basadas en el esqueleto que de forma eficiente modelen el refinamiento y desrefinamiento por bisección de lado mayor en 2D y 3D.

2. Estudio de las propiedades geométricas del grafo basado en el esqueleto y de la partición 4T-LE.

3. Implementación de los algoritmos de refinamiento y desrefinamiento basados en el esqueleto en 2D según las nuevas estructuras de datos propuestas. Análisis de las nuevas versiones de los algoritmos en comparación con los anteriores:

4. Presentar aplicaciones de los algoritmos a:

   a) Modelos Digitales del Terreno.

   b) Niveles de Detalle en Visualización.

   c) Incorporación de los algoritmos a un código de elementos finitos comercial y resolución de un problema no lineal.

Capítulo 2

Particiones y algoritmos

En algoritmos que tratan con entidades geométricas, como los algoritmos de refinamiento y desrefinamiento, es importante estudiar en detalle las transformaciones geométricas, en nuestro caso, la partición de elementos de una malla. En este capítulo se hace una revisión de las distintas formas de subdividir elementos en una malla y de los algoritmos de refinamiento que surgen de las particiones. Además, se estudian las particiones 4T-LE y 8T-LE así como los algoritmos basados en el esqueleto, por ser de interés en esta tesis. Finalmente se procede a estudiar el comportamiento asintótico de las particiones, proporcionando datos y resultados usados en los restantes capítulos.

2.1. Definiciones y preliminares

Introduciremos las siguientes definiciones y notaciones procedente de geometría y geometría computacional que serán usadas en adelante.

Definición 2.1.1 (*m*-simplex) *Sea* $V = \{X_0, X_1, \ldots, X_m\}$ *un conjunto de* $m+1$ *puntos en* $\mathbb{R}^n$ $(1 \leq m \leq n)$ *tal que* $\{\overrightarrow{X_0X_i}\colon 1 \leq i \leq m\}$ *es un conjunto de vectores linealmente independientes en* $\mathbb{R}^n$*. Entonces la envoltura convexa cerrada de* V *denotada por* $S =< V >=< X_0, X_1, \ldots, X_m >$ *se denominará un simplex m-dimensional o un m-simplex en* $\mathbb{R}^n$*, mientras que los puntos* $X_0, \ldots, X_m$ *se denominarán* vértices de S *[43].*

Nótese que el conjunto S puede también definirse de forma explícita como:

$$S = \{x \in \mathbb{R}^n \ / \ x = \textstyle\sum_{i=0}^{m} \lambda_i x_i \text{ con } \sum_{i=0}^{m} \lambda_i = 1 \text{ y } 0 \leq \lambda_i \leq 1 \text{ para } 0 \leq i \leq m\}$$

A lo largo de esta tesis se usarán los términos simplex y símplice indistintamente.

De esta manera un tetraedro (cerrado) se define por el conjunto (no ordenado) de sus vértices $< X_0, X_1, X_2, X_3 >$, y una cara por sus tres vértices $< X_i, X_j, X_k >$. Una arista queda definida por un par de vértices distintos $< X_i, X_j >$. Vértices, aristas, caras y tetraedros son, respectivamente $0-, 1-, 2-$, y $3-$símplices. Cualquier i-simplex contenido en un n-simplex S (con $i < n$) será denominado i-simplex ó i-cara de S. Como no importa el orden de los vértices para definir las caras, resulta que el número de k-caras de un m-símplice es igual a $\binom{m+1}{k+1}$. Así por ejemplo el número de $(m-1)$-caras es $\binom{m+1}{m} = \binom{m+1}{1} = m+1$.

Definición 2.1.2 (Malla conforme de símplices) *Sea Ω un conjunto acotado en $\mathbb{R}^n$ (n=2 o 3, aunque la definición es válida en general) con interior no vacío, $\overset{\circ}{\Omega} \neq \emptyset$, y de frontera poligonal $\partial\Omega$. Una partición de Ω en $n-$símplices $\tau = \{t_1, \ldots, t_n\}$ tal que (i) $\Omega = \cup t_i$; (ii) $\overset{\circ}{t_i} \cap \overset{\circ}{t_j} = \emptyset$ si $i \neq j$; (iii) $\overset{\circ}{t_i} \neq \emptyset$, se denominará una* **triangulación** *de Ω.*

Dos símplices t_i, t_j de τ se dicen **adyacentes** *si $t_i \cap t_j \neq \emptyset$.*

Si τ es tal que símplices adyacentes comparten una cara entera, o una arista entera o un vértice, se dice que τ es una **triangulación conforme** *o también que no hay nodos no conformes en τ (en dimensión tres se dirá una teselación conforme o también una triangulación conforme en dimensión tres), ver figura 2.1.*

Dos triangulaciones (conformes) τ y τ^* de un mismo conjunto acotado Ω se llamarán **encajadas** *o* **anidadas**, *y escribiremos $\tau < \tau^*$ si se cumple la siguiente condición: $\forall\, t \in \tau,\ \exists\, t_1,\ \ldots\ , t_p \in \tau^*$ tal que $t = t_1 \cup \ldots \cup t_p$. Diremos también que τ es más grosera que τ^*, o que τ^* es más fina que τ.*

Definición 2.1.3 (Esqueleto) *Sea τ una malla de n-símplices. El conjunto $skt(\tau) = \{f : f$ es una $(n-1)$-cara de algún t_i, con $t_i \in \tau\}$ se denominará* esqueleto *o $(n-1)$-esqueleto de τ, también denotado por $(n-1)$-skt(τ) [9].*

Por ejemplo, el esqueleto de una triangulación en dimensión tres está compuesto por las caras triangulares de los tetraedros, y en dimensión dos el esqueleto es el conjunto de las aristas de los triángulos. Es de destacar por otra

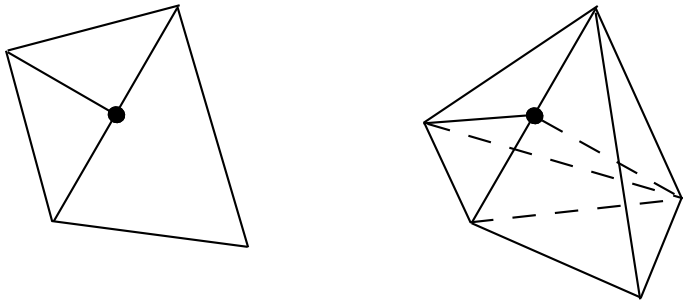


Figura 2.1: Nodo no conforme en dos y tres dimensiones

parte, que el esqueleto puede ser visto como una nueva triangulación del conjunto compuesto de todas las $(n-1)$-caras de la triangulación inicial. Nótese que el concepto de una malla conforme de símplices es aplicable al $skt(\tau)$ por estar compuesto por $(n-1)$-símplices. Notamos aquí que si τ es una triangulación conforme tridimensional en $\mathbb{R}^3$, $skt(\tau)$ es una triangulación conforme bidimensional contenida en $\mathbb{R}^3$ o una triangulación $2D\frac{1}{2}$ [27].

La definición anterior de esqueleto puede ser aplicada de forma recursiva. Es decir el conjunto: $skt(skt(\tau)) = \{f : f$ es una $(n-2)$-cara de alguna $s \in skt(\tau)\}$. De hecho, si τ es una malla n-simplicial, $skt(skt(\tau))$ se denominará el $(n-2)$-*esqueleto* de τ, y así sucesivamente.

Puntualizamos que este concepto no está relacionado con el de *objeto medio* también llamado esqueleto, usado en la literatura [37, 77], pero sí con la idea topológica de esqueleto de un complejo euclídeo [56].

Lema 2.1.1 *Si τ es conforme, $skt(\tau)$ es también conforme [63].* □

Definición 2.1.4 (Bisección por una arista) *Sea $S =< X_0, X_1, \ldots, X_m >$ un* m*-simplex en $\mathbb{R}^n$, con arista $< X_j, X_k >$ teniendo como punto medio $A = (X_j + X_k)/2$. Entonces podemos formar dos nuevos símplices*

$$S_1 =< X_0, \ldots, X_{j-1}, A, X_{j+1}, \ldots X_k \ldots X_m >$$
$$S_2 =< X_0, \ldots, X_j, \ldots, X_{k-1}, A, X_{k+1}, \ldots X_m >$$

Estos símplices tienen interiores disjuntos y $S = S_1 \cup S_2$. Esto define una subdivisión de S por bisección de la arista o bisección simple [43, 79].

Frecuentemente se escoge $< X_j, X_k >$ como la arista de bisección de S la arista mayor [9, 48, 83], (se dice entonces que se ha realizado una bisección por la arista mayor o bisección generalizada).

Definición 2.1.5 (Arista entorno) *Sea τ una malla simplicial n-dimensional. Al aplicar el refinamiento por medio de la bisección de los elementos, denominamos* arista entorno *de un nodo N a la arista en la cual N está en el punto medio.*

Esta arista también se puede llamar arista de generación de N como hacen Rivara y Plaza en [88].

Definición 2.1.6 (Envoltura de la arista) *Para cada arista E llamamos* envoltura de E*, y la denotamos por $h(E)$, al conjunto de símplices que están compartiendo la arista E: $h(E) = \{S \in \tau \ : \ E \in S\}$.*

En general la envoltura $h(E)$ de una arista relativa a la triangulación no es un conjunto convexo. En la figura 2.2 se dibujan la arista entorno de un nodo (en negro) y la envoltura de su arista entorno.

Definición 2.1.7 (Molécula de un nodo) *Si la arista tiene un nodo en su punto medio, como en el caso de la figura 2.2, el conjunto de* nodos extremos *de la envoltura de una arista (que son los puntos blancos en la figura) se denominará* molécula *de ese nodo.*

Obsérvese que esta definición puede ser extendida a cualquier dimensión.

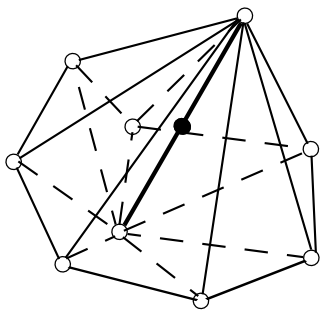


Figura 2.2: Envoltura relativa a la teselación

Lema 2.1.2 *La bisección de un m-simplex S por su arista mayor o por alguna arista seleccionada, induce la bisección de todos los k-símplices en S que contienen la arista seleccionada con $k < m$ [63].* □

Observación: En dimensión dos, al dividir un triángulo por el lado mayor aparece un nuevo nodo y una nueva arista. En dimensión tres aparece una nueva cara interna y dos nuevas aristas internas a las caras triangulares que conectan el nuevo vértice con los extremos de la arista opuesta, como se ve en la figura 2.3.

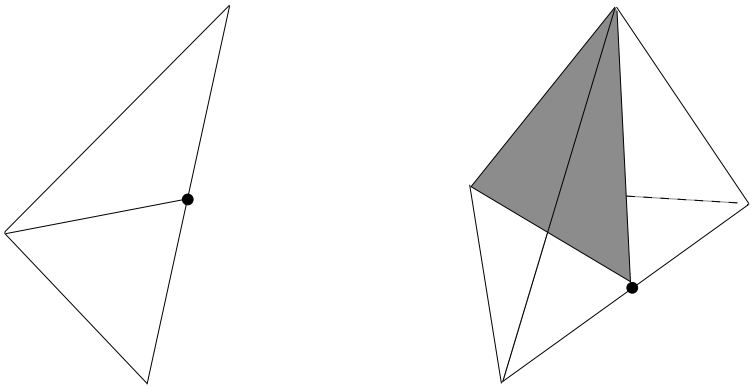


Figura 2.3: Bisección de un símplice: casos bi y tridimensional

Definición 2.1.8 (Topología) *Topología en un conjunto $\mathbb{X}$ es un sistema de subconjuntos de $\mathbb{X}$, $\mathcal{X}$ con las siguientes propiedades:*

1. $\emptyset, \mathbb{X} \in \mathcal{X}$
2. *si $\mathcal{Z} \subseteq \mathcal{X}$ entonces $\cup\mathcal{Z} \in \mathcal{X}$ y*
3. *si $\mathcal{Z} \subseteq \mathcal{X}$ es finito entonces $\cap\mathcal{Z} \in \mathcal{X}$.*

Un conjunto $\mathbb{X}$ con una topología se llama espacio topológico y se escribe $(\mathbb{X}, \mathcal{X})$.

Un *disco abierto* de un punto en $\mathbb{X} \subset \mathbb{R}^2$ es una porción del espacio bidimensional que cae dentro de algún círculo con radio positivo centrado en el punto dado y que excluye el propio círculo. Una *bola abierta* es la equivalencia de disco abierto en el espacio 3D. Un subconjunto de un espacio topológico

es *arco-conectado* si entre dos puntos cualesquiera del subconjunto existe un camino continuo que esta completamente contenido dentro del subconjunto del espacio. Una *superficie* para nuestros propósitos es un espacio bidimensional arco-conectado. Hay que notar que aunque una superficie es localmente bidimensional, puede existir geométricamente en un espacio tridimensional y ésta puede ser curvada. A menudo se nombra este espacio donde residen las superficies, espacio $2\frac{1}{2}D$. Una superficie es *acotada* si la superficie completa puede ser contenida en alguna bola abierta. *Contorno o borde* de la superficie puede ser una curva abierta o cerrada, o un punto de la superficie. Un contorno cerrado separa una parte de la superficie del resto de la misma. Una superficie es *cerrada* si es acotada y no tiene contorno. Por ejemplo, un plano no tiene contorno pero es no acotado, mientras que una esfera es una superficie cerrada.

Figura 2.4: Relaciones de adyacencia posibles para las entidades de vértices (V), caras (F) y aristas (E)

Un *campo* (manifold), en particular un 2-campo es una superficie conectada en 2D donde cada punto de la superficie tiene un entorno topológicamente equivalente a un disco abierto. Un campo puede ser una superficie cerrada o no. Un campo es *orientable* si tiene dos lados perfectamente diferenciados. Por ejemplo la botella de Klein no es orientable. Las superficies de un volumen sólido se requiere que sean orientables y cerradas, de tal forma que haya una clara distinción entre el interior y exterior del volumen.

Las adyacencias topológicas conciernen a las adyacencias físicas entre las entidades de una malla. Entendemos por entidad física una primitiva topológica del esqueleto de una malla. Para superficies distinguimos vértice, arista y cara. Al igual que definimos en 2.1.2 la adyacencia entre m-símplices, una relación de adyacencia entre las primitivas topológicas indica la adyacencia topológica de un grupo de elementos de un tipo específico en relación a un elemento del tipo especificado. Las relaciones de adyacencia contienen dos tipos de información: la clase de adyacencia que representa y el orden de los elementos adyacentes. En la figura 2.4 se representan las nueve relaciones de adyacencia topológica entre caras, aristas y vértices para un tetraedro de ejemplo.

2.2. Algoritmos de refinamiento y particiones en 2D

Un gran número de algoritmos de refinamiento y particiones han sido desarrollados en los últimos años. Especial interés ha tenido el refinamiento de elementos simpliciales, triángulos en 2D y tetraedros en 3D. En esta sección comenzamos revisando algunos de los más importantes en 2D.

Definición 2.2.1 (Partición Similar 4T) *El triángulo original se divide en cuatro triángulos conectando los puntos medios de las aristas con segmentos rectos.*

Se obtienen así cuatro triángulos que son semejantes al original (ver figura 2.5 (a)). La partición Similar 4T es una de las más simples y usadas en la literatura. Carey 1976 [12] fue el primero en aplicar esta división en el ámbito del Refinamiento Adaptativo de Mallas (AMR). Bank y Sherman [5] también han desarrollado un algoritmo basado en este esquema.

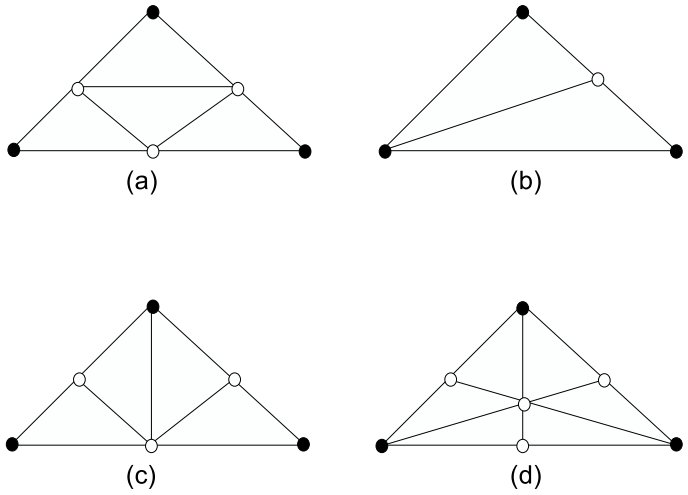


Figura 2.5: Diversas formas de dividir un triángulo
(a) Partición Similar 4T, (b) Bisección, (c) Partición 4T-LE,
(d) Partición Baricéntrica

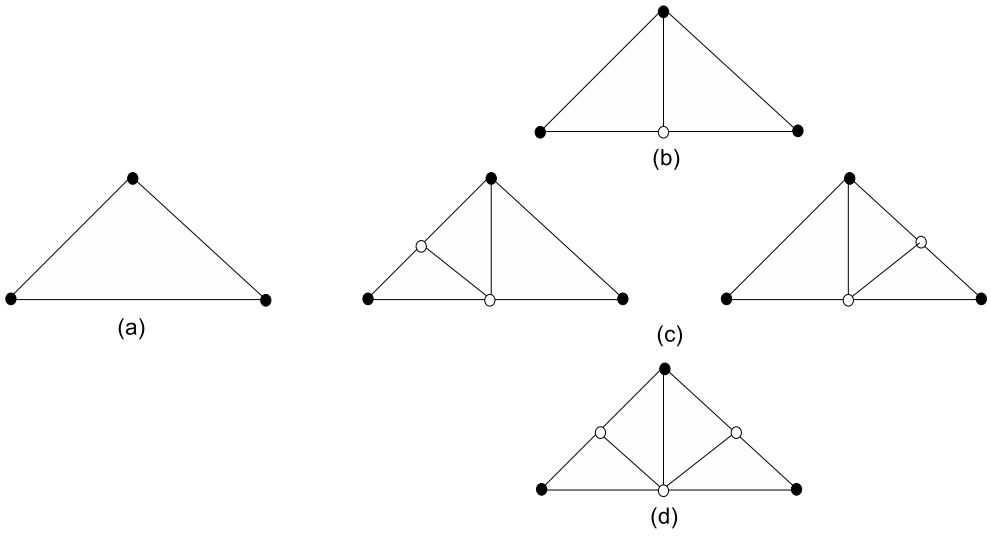


Figura 2.6: Las cuatro posibilidades de división 4T
(a) triángulo original, (b) división en dos triángulos, (c) división en tres, y
(d) división en cuatro

El algoritmo desarrollado por Bank y Sherman [5] (ver figura 2.5 (a)) subdivide un triángulo como se ha dicho antes. Se mantiene la regularidad en cuanto a los ángulos. Por tanto, si realizamos refinamientos globales mediante este algoritmo y representamos por τ_0 la triangulación inicial y por τ_n las obtenidas tras n refinamientos globales, se tendrá que $\alpha(\tau_0) = \alpha(\tau_n)$, con lo que se conserva el ángulo mínimo, puesto que todos los triángulos generados son semejantes al original. Sin embargo, si se refina localmente, con objeto de asegurar la continuidad de la solución, Bank introduce la denominada *green division* (ver figura 2.7) en la que los nodos que se encuentran en el punto medio de alguna cara de algún triángulo de la malla más fina, es decir, los llamados *nodos no conformes*, se unen al vértice opuesto o a otro vértice no conforme mediante la inclusión de una arista. Esta estrategia tiene el inconveniente de que los triángulos así divididos pierden las buenas propiedades de los divididos regularmente respecto de su forma, aunque por otra parte, tiene la ventaja de que la conformidad se logra sin añadir ningún nodo adicional.

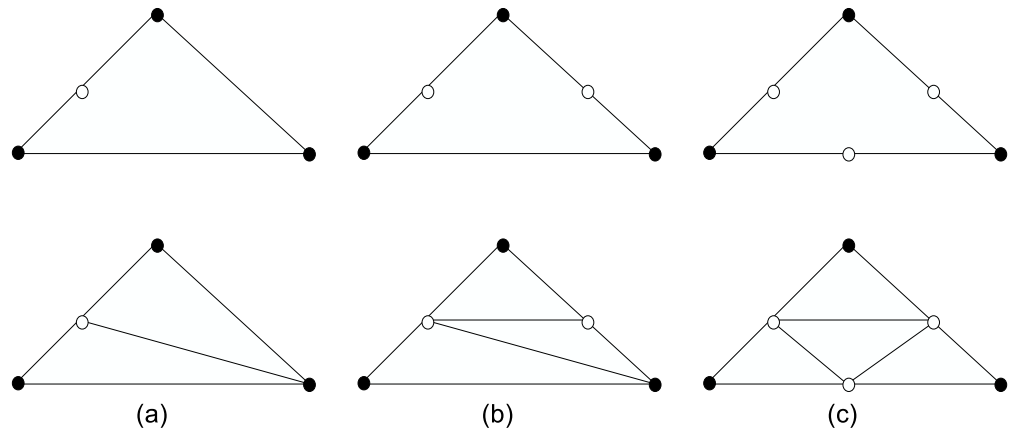


Figura 2.7: *"Green division"* para alcanzar la conformidad (a), (b), y (c), según tengamos 1, 2 o 3 nodos no-conformes

Definición 2.2.2 (Bisección simple y bisección LE) *La bisección simple consiste en dividir el triángulo en dos sub-triángulos uniendo el punto medio de uno de sus lados con su vértice opuesto (ver figura 2.5 (b)). Cuando el lado a dividir es el lado mayor del triángulo tenemos la bisección por el lado mayor, en lo que sigue, bisección LE (Longest Edge).*

Si consideramos la bisección simple, se tiene el problema de que, en general la reiteración de esta forma de refinamiento hace que el ángulo mínimo de la malla tienda rápidamente a cero. Aparecen triángulos degenerados que se comportan mal numéricamente. En este sentido, la bisección de un triángulo se puede mejorar, mediante la bisección por el lado mayor (LE). Se tiene entonces el algoritmo 2-T de Rivara o bisección generalizada.

Definición 2.2.3 (Partición 4T-LE) *La partición 4-T LE consiste en dividir un triángulo en cuatro del siguiente modo (ver figura 2.5 (c)): en primer lugar se divide el triángulo inicial mediante bisección por el lado mayor (bisección generalizada) y después mediante segmentos paralelos a los otros dos lados por el punto medio.*

Rivara [79, 80, 82, 85, 87] ha desarrollado y estudiado un algoritmo eficiente de bisección basado en la partición 4T-LE. En la figura 2.6 se presentan las cuatro divisiones que pueden generarse al aplicar un refinamiento basado en la partición 4T-LE. Tanto la bisección generalizada, como el algoritmo 4T-LE tienen muy buenas propiedades en cuanto a la regularidad de los elementos a los que da lugar. Un algoritmo equivalente basado en el esqueleto de una triangulación ha sido desarrollado por Plaza y Carey en [71]. La conformidad en los algoritmos 2-T y 4-T se logra del siguiente modo: si algún triángulo tiene algún nodo no conforme en uno de los lados que no sea el mayor se introduce otro nodo en el lado mayor y se une con el vértice opuesto; después un segmento paralelo a uno de los lados hace conforme el nodo inicial. Este proceso puede extenderse a otros triángulos vecinos. En este caso se dice que el refinamiento se extiende por conformidad. En estos algoritmos, los ángulos de los triángulos para refinamientos de mallas locales están uniformemente acotados entre 0 y π.

Otro método de bisección es el presentado por Mitchell [54]. La arista del triángulo que va a ser dividida se determina sin ningún tipo de cálculo computacional. Solamente se crean cuatro clases de triángulos similares y ocho ángulos distintos. Se satisface también la importante condición de que los ángulos están limitados entre 0 y π. También necesita refinamientos adicionales para conseguir la conformidad de la malla. Es de destacar que el algoritmo de Mitchell equivale al 4T de Rivara si la forma en que se elige la arista de bisección (es decir, la arista por la que se realiza la primera bisección), toma siempre

la arista mayor del triángulo.

Definición 2.2.4 (Partición Baricéntrica) *Para cualquier triángulo, la partición Baricéntrica consiste en conectar mediante un segmento recto los vértices y los puntos medios de los lados con el baricentro del triángulo, ver figura 2.5 (d).*

2.3. Algoritmos de refinamiento y particiones en 3D

En lo que sigue se presentan diversas definiciones que corresponden a particiones de tetraedros en dimensión 3. Para una mejor apreciación gráfica de las particiones se ha optado por representar no sólo la vista en 3D sino además una vista auxiliar análoga a la de Bänsch [7], que surge al desarrollar las 4 caras del tetraedro en el plano y transformar dichas caras a triángulos equiláteros.

Definición 2.3.1 (Partición Similar) *El tetraedro original es dividido en ocho subtetraedros cortando las cuatro esquinas por planos paralelos a la cara opuesta y dividiendo el octaedro interior resultante en cuatro subtetraedros más, ver figura 2.8.*

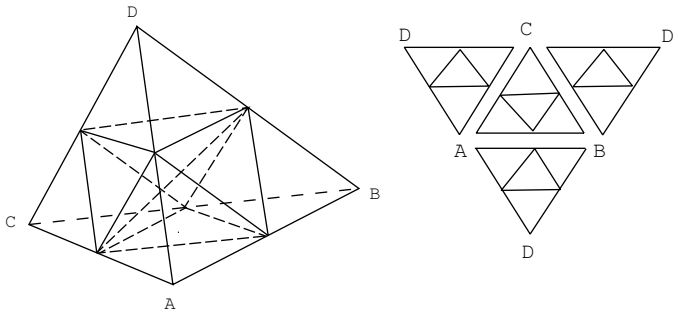


Figura 2.8: Partición similar en 3D y vista coplanaria de las caras

En la división del octaedro interno en la partición similar, es relevante la elección de la arista interna, ya que ello permite optimizar la forma de los cuatro tetraedros que se generan en su interior. Aunque llamaremos a esta partición partición similar, hay que señalar que sólo los cuatro tetraedros hijos que están en los vértices del tetraedro original son semejantes a él.

Definición 2.3.2 (Partición 8T-LE) *Para un tetraedro t con arista mayor única, la partición 8T-LE (8 Tetrahedra Longest-Edge) de t se define así:*

1. *Bisección por la arista mayor de t produciendo tetraedros t_1, t_2.*
2. *Bisección de t_i por el punto medio de la única arista de t_i que es arista mayor de la cara común con el tetraedro original t, produciendo tetraedros t_{ij}, para $i, j = 1, 2$.*
3. *Finalmente, bisección de cada t_{ij} por el punto medio de la única arista común con el tetraedro original t.*

La figura 2.9 muestra uno de los posibles casos al aplicar la partición 8T-LE. Más adelante, en este mismo capítulo, se estudiarán todos los casos posibles.

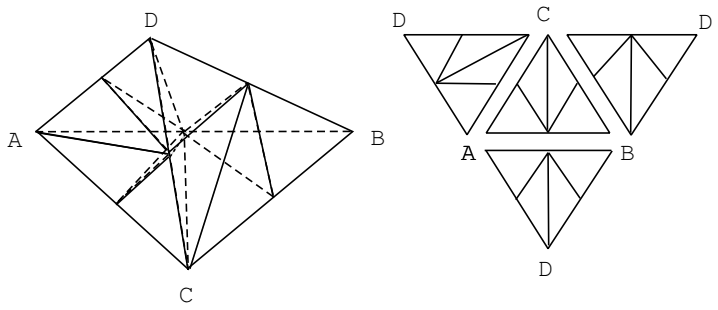


Figura 2.9: Caso ilustrativo de la partición 8T-LE

Definición 2.3.3 (Partición 3D-Baricéntrica) *Para un tetraedro t la partición baricéntrica se define así, ver figura 2.10:*

1. *Añadir un nodo nuevo P en el baricentro de t y en los baricentros de las caras y aristas de t.*
2. *En cada cara de t aplicar la partición baricéntrica 2D de la cara.*
3. *Unir el baricentro P con todos los vértices de t y con todos los nuevos nodos añadidos anteriormente.*

Un algoritmo de refinamiento basado en la partición similar es el de Bey [10]. Para refinamientos irregulares a la hora de alcanzar la conformidad, Bey considera cuatro patrones que cubren 25 de los 64 posibles, como se ve en la

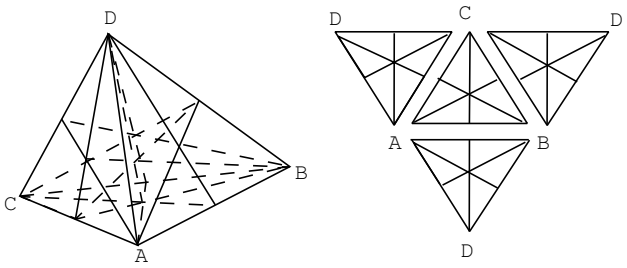


Figura 2.10: Partición baricéntrica 3D

figura 2.11 (nótese que hay $2^6 = 64$ posibilidades, porque sólo considera el número de nuevos nodos que aparecen en los puntos medios de las aristas, y no su posición relativa de acuerdo con la longitud de las mismas). Para el resto de los casos realiza refinamientos globales. Esto implica que en determinados casos puede producirse un *efecto dominó*, en el sentido de tener una propagación excesiva del refinamiento a través de la malla.

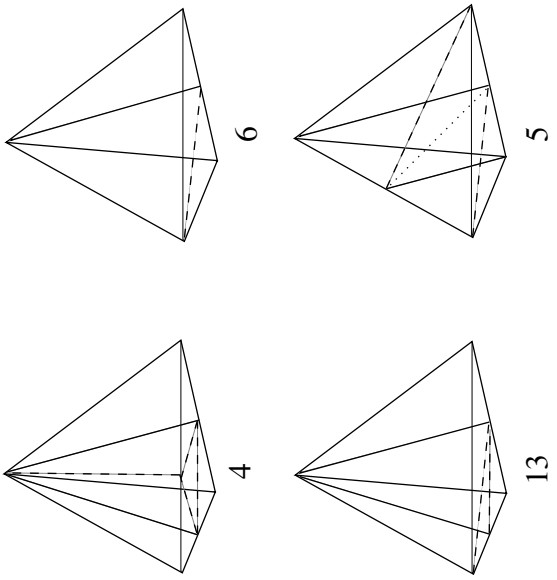
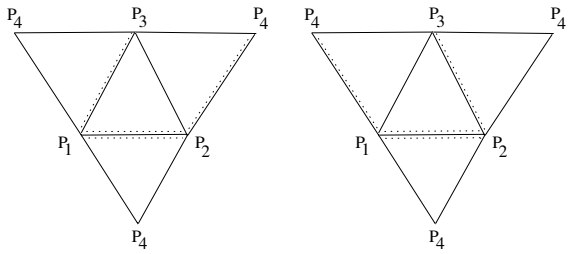


Figura 2.11: Patrones usados por Bey que cubren $(4 + 6 + 12 + 3) = 25$ posibilidades

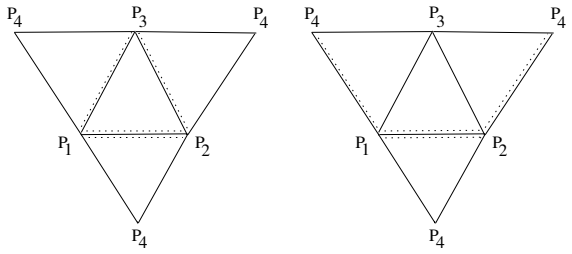
E. Bänsch [7] presenta un algoritmo basado en la elección de una arista como *arista de refinamiento global* en cada tetraedro, pero debe imponer pequeñas

perturbaciones en las coordenadas de los nodos para evitar ciertas incompatibilidades. Él clasifica los tetraedros en dos tipos, denominados *rojo* y *negro*, dependiendo de la posición relativa de las aristas de división y para ello al principio los abre sobre el plano.

En la figura 2.12 se muestran los tetraedros tipo *rojo* y *negro* abiertos, la arista de refinamiento global con doble líneas de puntos y las siguientes en el orden de bisección con una simple línea de puntos.



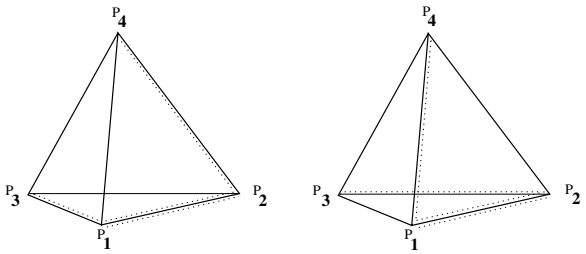
(a) Rojo



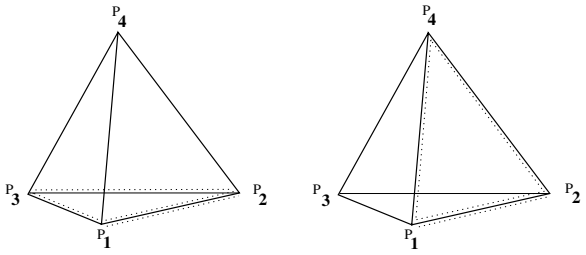
(b) Negro

Figura 2.12: Tetraedros abiertos clasificados como rojo y negro

Una vez cerrados los tetraedros, ver la figura 2.13, se observan más claramente los tipo rojo y negro, aunque los dos tetraedros que dan cuenta de los tetraedros de tipo rojo son equivalentes bajo una rotación de π radianes a través de un eje que pase por los puntos medios de las aristas con nodos P_1 y P_2 y la opuesta con nodos P_3 y P_4.



(a) Tipo rojo



(b) Tipo negro

Figura 2.13: Tetraedros tipo rojo y negro

Algoritmos de refinamiento de mallas de tetraedros basados en la bisección son los de Muthukrishnan *et al.* [59] y Rivara y Levin [83]. Ambos dividen los tetraedros por bisección por el lado mayor. Para alcanzar la conformmidad, también se aplica bisección por el lado mayor de tetraedros adyacentes. Aunque no se realiza un estudio teórico-matemático sobre la no degeneración de las mallas creadas por medio de este método, los datos experimentales aportados dan a entender la buena calidad de las mallas obtenidas por medio de este algoritmo, y que incluso se tiene una cierta propiedad de mejoramiento de la malla cuando se parte de una malla inicial muy mala.

Recientemente Plaza y Carey [71] han presentado una generalización del algoritmo 4T de Rivara a tres dimensiones. Esta extensión ha sido llamada algoritmo 3D-SBR (3D-Skeleton Based Refinement). Algunas propiedades geométricas de este algoritmo han sido descritas por Rivara y Plaza [88]. El algoritmo trabaja primero sobre las caras triangulares de los tetraedros, el 2-esqueleto de la teselación 3D y después subdivide el interior de cada tetraedro de forma congruente con la subdivisión del esqueleto. Esta idea podría ser

aplicada para desarrollar algoritmos semejantes en dimensiones mayores. El algoritmo se puede utilizar para cualquier malla inicial de tetraedros sin ningún preproceso. Esta es una propiedad importante con relación a otros algoritmos similares. Como en el refinamiento, también el desrefinamiento, el algoritmo inverso, está basado en el 2-esqueleto y utiliza los mismos patrones de elementos usados al refinar. Ambos algoritmos son de complejidad lineal y han sido aplicados a diversos problemas [63, 72].

Liu y Joe [47, 48] presentan un algoritmo similar al de Bänsch (QLRB: *Quality Local Refinement Based algorithm*). Ellos clasifican los tetraedros en cuatro tipos y clasifican los tipos de aristas dependiendo del tipo de tetraedro, como se ve en la figura 2.14.

La clasificación de los tetraedros en cuatro tipos se realiza de la forma siguiente:

1. **DD**: Aparecen aristas marcadas 1 y 2 en aristas opuestas. Siendo la arista etiquetada 1 la arista de referencia.

2. **DSS1**: La arista opuesta a la arista de referencia del tetraedro es etiquetada 3 y las aristas etiquetadas 2 se encuentran en la misma cara.

3. **DSS2**: Las aristas tipo 2 se encuentran en aristas opuestas.

4. **DSS3**: Las aristas tipo 2 se encuentran en la misma cara pero una de ellas es opuesta a la arista de referencia.

También presentan una medida de la calidad de los tetraedros obtenidos. Se prueba también que el número de clases de semejanza es acotado de forma que las mallas no pueden degenerar.

Se han desarrollado otros estudios similares a los de Bänsch y Liu y Joe. Kossaczký [45] desarrolla una aproximación recursiva. Su algoritmo necesita imponer ciertas restricciones y hacer un preproceso a la malla inicial. Su propuesta es equivalente a las de Bänsch y Liu y Joe en el sentido de que produce las mismas mallas al refinar. Por otro lado Maubach [53] desarrolla un algoritmo para mallas de n-símplices generadas por reflexión. Aunque el algoritmo es válido para cualquier dimensión y el número de casos de semejanza es acotado,

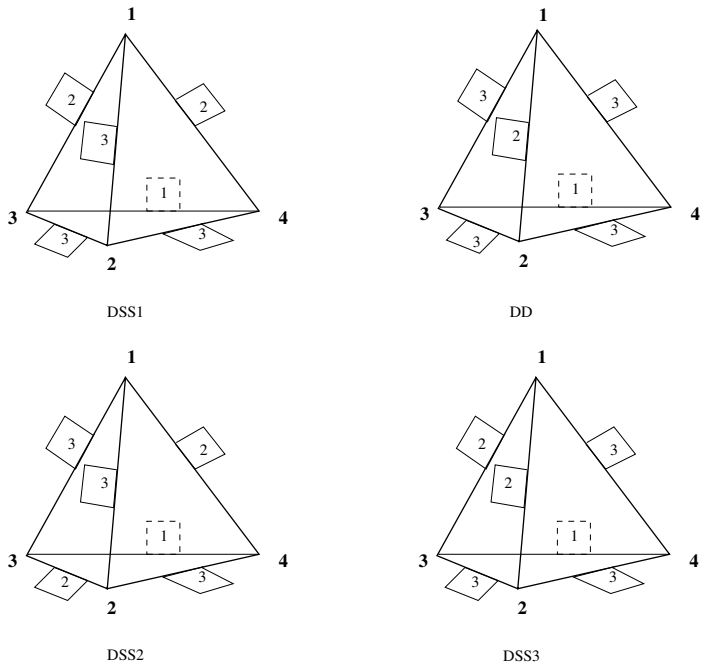


Figura 2.14: Clasificación de los tetraedros según Liu y Joe

no puede ser aplicado a cualquier malla inicial de tetraedros. Es necesario un cierto refinamiento adicional para evitar ciertas incompatibilidades. Recientemente Mukherjee [2, 57] ha presentado un algoritmo equivalente al de Bänsch y prueba la equivalencia con el de Maubach.

2.4. Las particiones 4T-LE y 8T-LE

Centramos ahora la atención en las particiones 4T-LE y 8T-LE para la descripción de los algoritmos de refinamiento en dimensión dos y tres. Ambos algoritmos están basados en la bisección y también ambos realizan una clasificación de las aristas que definirán las sucesivas bisecciones. Teniendo en cuenta la definición 2.1.7 y el lema 2.1.2, si se conocen las aristas de bisección, y el orden en que se toman para subdividir el tetraedro, entonces la subdivisión está perfectamente definida. En este caso el orden está basado en la clasificación de las aristas según su longitud.

El Algoritmo 4-T desarrollado por Rivara en 1984 [79, 80, 81] como un tipo particular de algoritmo basado en la bisección por la arista mayor, se puede

describir en función de tres patrones de refinamiento que aparecen en la figura 2.15, donde P es el punto medio de la arista mayor.

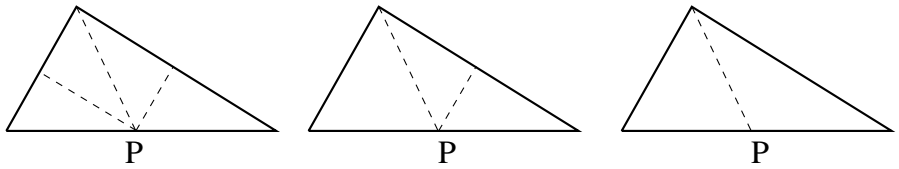


Figura 2.15: Patrones del refinamiento 4-T

Un ejemplo de aplicación del algoritmo 4-T para el refinamiento local de un triángulo t, aparece en la figura 2.16. En el caso más general, el triángulo t puede tener triángulos vecinos que comparten las aristas AC y BC y en tal caso, los pasos de propagación por conformidad se tienen que aplicar de forma consistente.

El algoritmo, según Rivara [79, 80, 81] que refina un triángulo individual t de una triangulación conforme τ se expone en el siguiente pseudocódigo:

El Algoritmo 4-T de Refinamiento $\{\tau, t\}$
/* Se realiza la división de t */
Para cada arista e de t, con vecino asociado $t^\star$, **hace**
    refinamiento-vecino $(t^\star, e)$
    $t \leftarrow t^\star$
    Mientras t es no conforme, **hace**
        encuentra la arista no conforme e de t con vecino $t^\star$
        Refinamiento-vecino $(t^\star, e)$
        $t \leftarrow t^\star$
    Fin del mientras
Fin del para

refinamiento-vecino $(t^\star, e)$
Si e es la arista mayor de t, **entonces**
    realiza la bisección por la arista mayor de t
De lo contrario

realiza la división en tres del triángulo t por la arista e
Fin del si

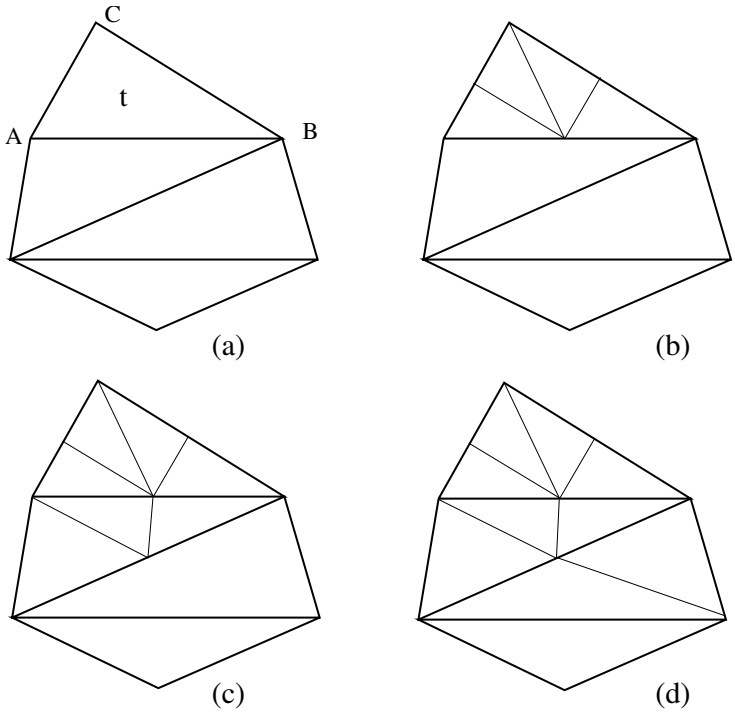


Figura 2.16: Ejemplo del algoritmo de refinamiento 4-T

La aplicación del algoritmo 4-T produce un subconjunto de triángulos que se obtienen a partir de la bisección de la arista mayor, de forma que podemos enunciar el siguiente teorema basado en las propiedades de la división reiterada por la arista mayor de los triángulos [81].

Teorema 2.4.1 *La aplicación reiterada del algoritmo 4-T sobre cualquier triangulación conforme τ: (1) produce triángulos cuyo ángulo mínimo está acotado como función de la calidad inicial de la triangulación; (2) produce un número finito de clases de semejanza; (3) las triangulaciones que se obtienen tienden a mejorar en el sentido de que el porcentaje de los triángulos de buena calidad y el área cubierta por ellos, se incrementa conforme se realiza el refinamiento.* □

Se puede resaltar además, que la partición del algoritmo 4-T (también llamado 4T-LE) da lugar a las siguientes propiedades de mejora [79, 81, 84]:

Teorema 2.4.2 *Para cualquer triángulo obtuso t_0 cuyo ángulo más pequeño sea α_0 y ángulo mayor γ_0, la partición de t_0 da lugar a un único triángulo no semejante t_1, cuya división por el 4-T da lugar a un nuevo triángulo no semejante t_2, y así sucesivamente hasta que se obtiene el último triángulo no obtuso t_n. Por otra parte, los ángulos más pequeños α_i y los más grandes γ_i de cada t_i satisfacen las siguientes relaciones:*

$$\alpha_0 < \alpha_1 < \alpha_2 < \ldots < \alpha_n$$
$$\gamma_0 > \gamma_1 > \gamma_2 > \ldots > \gamma_n$$

□

Además, para el algoritmo 4T-LE, se puede probar fácilmente una propiedad fractal análoga a la que se da en el algoritmo de la bisección generalizada (bisección LE) [84, 85]. Para este fin, recordamos el concepto de *molécula* asociada a cualquier vértice, según Rivara:

Definición 2.4.1 *Para cualquier triangulación conforme τ, la molécula asociada a P es la lista ordenada de los ángulos de los triángulos que comparten el vértice P en τ.*

Teorema 2.4.3 *[85] Después de la aplicación iterativa (local) de un número finito de veces del algoritmo 4T-LE alrededor de cualquier vértice P de cualquier triangulación conforme τ, se obtiene una molécula estable alrededor de P, en el sentido de que las siguientes iteraciones del algoritmo no dividen los ángulos de la molécula estable, pues únicamente introducen nuevos vértices a lo largo de las aristas de la molécula estable. Además, cada nuevo triángulo en torno a P a través de las iteraciones, será semejante a su triángulo padre (comportamiento fractal).* □

En la figura 2.17 se observa un ejemplo de división 4T-LE y su comportamiento fractal.

A continuación se presentará extensamente la partición 8T-LE (que generaliza la partición 4-T de Rivara), según ha sido estudiada por Rivara y Plaza en [88]. Esta partición para cada tetraedro t produce la partición en 4-triángulos de las caras de t y una triangulación conforme en volumen de t tal que cada nuevo tetraedro en esta triangulación tiene al menos una *cara externa* en el tetraedro original t.

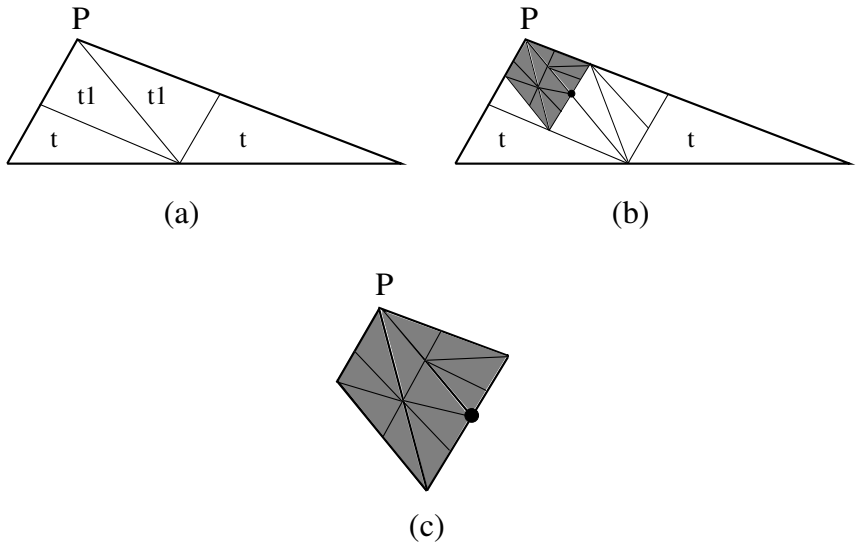


Figura 2.17: La división 4T-LE y su comportamiento fractal. La figura (c) muestra la molécula estable de P

Definición 2.4.2 *Para cada tetraedro t con arista mayor única, se llaman caras primarias de t las dos caras que comparten la arista mayor. Las otras dos caras serán llamadas caras secundarias. Además, la arista mayor de t se llama arista mayor primaria de t, o arista de referencia de t.*

Definición 2.4.3 *Para cada tetraedro t con arista mayor única, se llaman aristas mayores secundarias de t las aristas mayores de las caras secundarias de t (1 o 2 aristas mayores secundarias). Además, las 3 o 4 aristas restantes de t se llaman aristas de tercera clase.*

En la figura 2.18 se observa la posición de la arista mayor (numerada 1) y las aristas secundarias (numeradas 2) de un tetraedro t.

Se cumplen las siguientes propiedades [88]:

Proposición 2.4.1 *Para cada tetraedro t con arista mayor única, las caras primarias de t comparten una arista mayor común e igual a la arista mayor de t.*

Como ilustración de la propiedad anterior, véase la figura 2.19 en la que la única arista mayor AB de t es además la arista mayor común a las caras ADB y ACB (caras primarias de t).

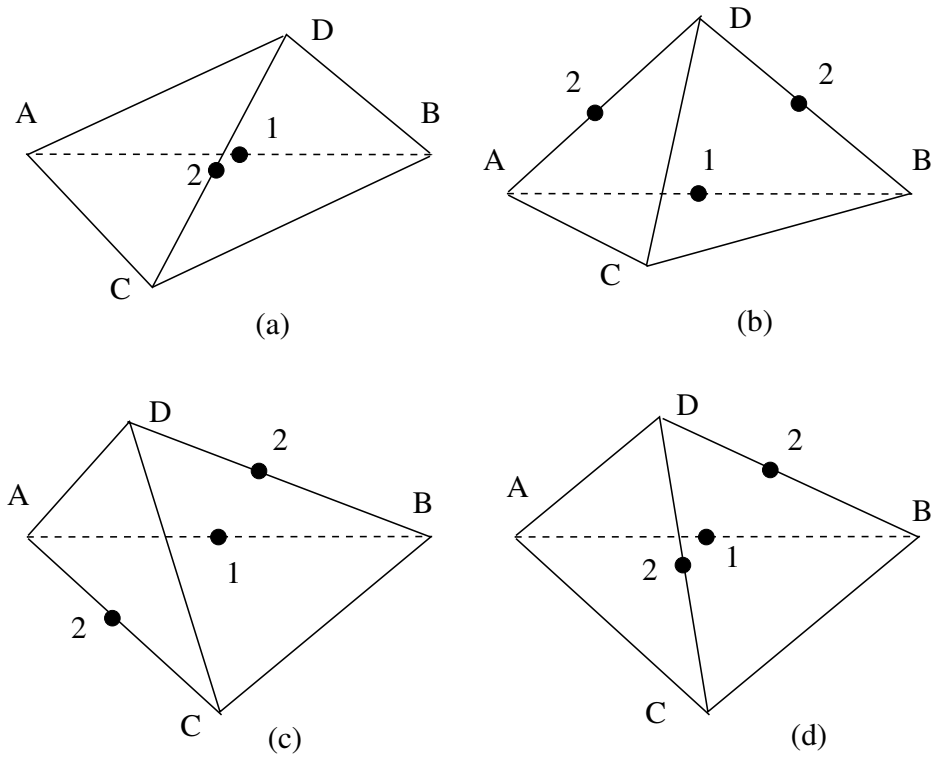


Figura 2.18: Posición relativa de la arista mayor (numerada 1) y de las aristas secundarias (numeradas 2) de t

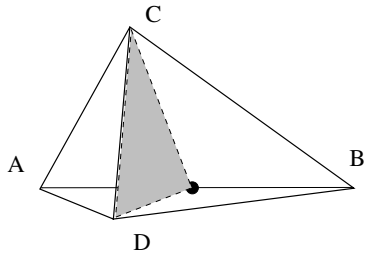


Figura 2.19: Partición de las dos caras que comparten la arista mayor.

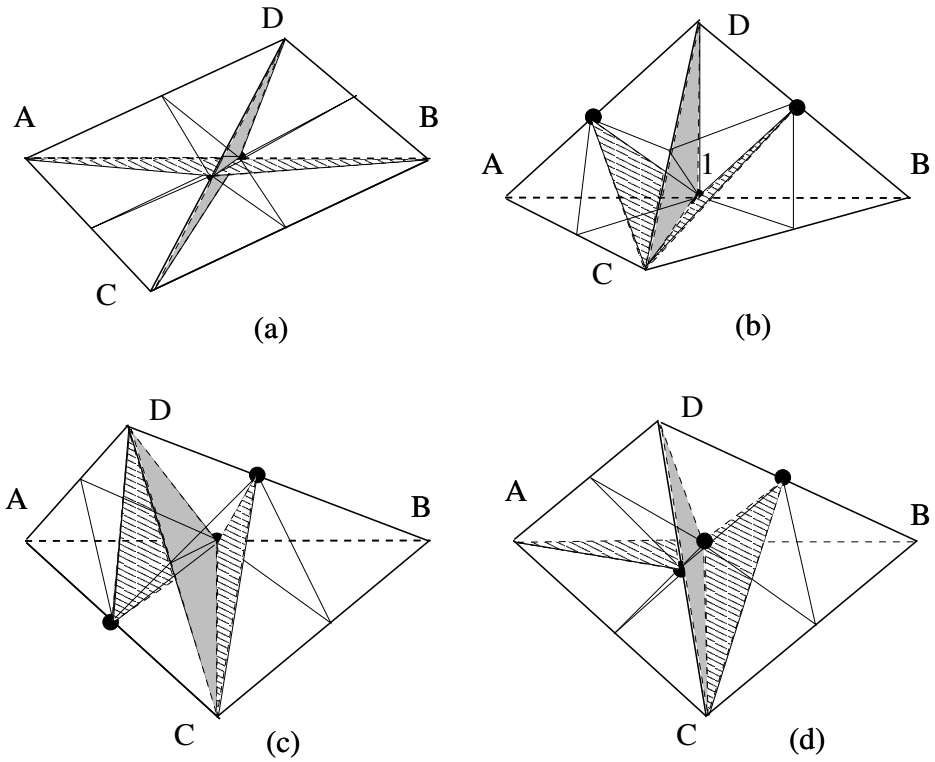


Figura 2.20: Posibles divisiones 8T-LE distintas de t

La partición 8T-LE en función de los tipos de aristas analizadas se puede formular explícitamente como sigue:

Definición 2.4.4 *Para cada tetraedro t con arista mayor única, la división 8T-LE de t se define como sigue:*

1. *bisección LE de t produciendo tetraedros $t_1,\ t_2$*
2. *bisección de t_i por el punto medio de la (única) arista de t_i que es además una arista secundaria de t, produciendo tetraedros t_{ij} para $i, j = 1, 2$*
3. *bisección de cada t_{ij} por el punto medio de la (única) arista igual a una arista de tercera o de segunda clase de t, para $i, j = 1, 2$.*

Teorema 2.4.4 *La partición 8T-LE de cualquier tetraedro t produce una triangulación conforme en volumen de t y una triangulación conforme de superficie de t tal que:*

1. *La triangulación conforme de superficie de t es idéntica a la triangulación de superficie obtenida mediante la división en 4-triángulos de las caras de t.*
2. *Se obtienen 4 patrones diferentes para la triangulación en volumen (ver figuras 2.18 y 2.20) de acuerdo con la posición relativa de la arista mayor y de las aristas secundarias de t.* □

Es de destacar la correspondencia biunívoca entre los cuatro patrones (en volumen) producidos por la división 8T-LE de un tetraedro t y los cuatro patrones de superficie obtenidos mediante la división en 4 triángulos de las caras de t. Esta propiedad nos permite formular el algoritmo de refinamiento 8T-LE (también llamado 3D-SBR) para una malla cualquiera τ en términos del refinamiento de la malla del esqueleto de τ.

La división 8T-LE generaliza la división 4-T de Rivara de la figura 2.15 (a). Nótese sin embargo que para formular de forma precisa el algoritmo en 3D, se necesita definir un conjunto de divisiones parciales válidas (análogas a las divisiones bi-dimensionales de la figura 2.15 (b) y 2.15 (c)). Estas particiones parciales serán empleadas en el paso de propagación por conformidad asociado

a esta particular implementación del algoritmo por el lado mayor (LE).

Un estudio cuidadoso de los posibles patrones de división con n puntos, producidos por diferentes posiciones relativas de la arista mayor y de las aristas secundarias de t, para $n = 1, 2, \ldots, 6$ (lo que incluye la división global 8T-LE) permite obtener los siguientes resultados [88]:

Teorema 2.4.5 *El refinamiento en volumen de la malla que se obtiene por la aplicación del 3D-SBR induce el refinamiento de la superficie del 2D-esqueleto de la malla. Además, la superficie de la malla refinada es idéntica a la malla que se obtiene por aplicación del algoritmo de refinamiento 4T a las caras de* t. □

Teorema 2.4.6 *Para cualquier malla de tetraedros conforme* τ_0*, después de un número finito de refinamientos locales por medio del 3D-SBR alrededor de algún vértice P, se obtiene un número finito de tetraedros que comparten el vértice P, cuyos ángulos sólidos asociados al vértice P ya no se vuelven a refinar más aunque el proceso de refinamiento continúe en torno a P (propiedad fractal).* □

Nótese que el teorema anterior es la versión en 3D de la propiedad análoga en 2D (teorema 2.4.3).

2.5. Los algoritmos 2D-SBR y 3D-SBR

A partir de la idea de esqueleto de una triangulación es posible formular un algoritmo general de refinamiento de símplices para dimensión n como sigue:

Algoritmo de Refinamiento (τ, *n-símplice, partición*)
Para cada *n-símplice* a refinar, **hacer**
1) División del *1-esqueleto*
2) División del *2-esqueleto*
...
n) Reconstrucción del *n-símplice*
Fin del para

El esquema anterior acepta como entrada la triangulación τ inicial, el n-símplice (o varios) a refinar y la *partición* usada para dividir los elementos. Como salida, el algoritmo produce la nueva triangulación τ. El algoritmo divide los esqueletos de la triangulación dada en orden inverso, hasta el $(n-1)$-esqueleto y finalmente procede con la reconstrucción del n-símplice. Este último paso precisa además de cierta información geométrica que se deriva de la partición, por ejemplo, creación de nodos internos, conexión entre caras y aristas etc.

En relación con el algoritmo de refinamiento 4T, una versión alternativa es el denominando 2D-SBR -algoritmo de refinamiento en 2D basado en el esqueleto-, desarrollado por Plaza y Carey [68, 71]. Dicho algoritmo se ajusta al esquema algorítmico basado en el esqueleto presentado al inicio. Básicamente realiza el refinamiento en dos pasos consecutivos de la siguiente forma: (1) Identifica y divide las aristas que intervienen a través de todo el proceso de refinamiento; y (2) divide cada uno de los triángulos en el proceso de refinamiento como se expone en la figura 2.15 (que depende del número de aristas divididas de cada triángulo).

El siguiente esquema resume el algoritmo 2D basado en el esqueleto. Nótese que si bien este esquema hace uso de la idea geométrica del esqueleto de una triangulación, esto no se refleja en la estructura de datos utilizada. Precisamente ése es uno de los objetivos de este trabajo como ya se ha comentado en la sección de objetivos 1.6.

El Algoritmo de Refinamiento basado en el Esqueleto (τ, t)
Encuentra y divide las aristas (τ, t)
División de los triángulos (τ, t)

Encuentra y divide las aristas (τ, t)
/* Divide las tres aristas de t */
Para cada arista e de t, **hacer**
    Encuentra el vecino $t^\star$ de t por la arista e
    Mientras la arista mayor de $e^\star$ de $t^\star$ sea distinta de e, **hacer**
        Divide la arista mayor $t^\star$ de $e^\star$
        $t \leftarrow t^\star$
        $e \leftarrow e^\star$
        Encuentra al vecino $t^\star$ de t por la arista e
    Fin del mientras
Fin del para

División de los triángulos (τ, t)
Para cada triángulo $t^\star$ que tiene al menos una arista dividida, **hacer**
    Si el triángulo $t^\star$ tiene una sola arista dividida, **entonces**
        Divide $t^\star$ por bisección de la arista mayor
    Si tiene dos aristas divididas, **entonces**
        Realiza la división en tres del triángulo $t^\star$
    Si no
        Realiza la división en cuatro de $t^\star$
    Fin del si
Fin del para

La figura 2.21 ilustra el uso de los dos pasos principales de este algoritmo para refinar la triangulación (a) de la figura 2.16.

En este sentido, remarcamos los siguientes puntos:

1. El procedimiento de bisección por la arista mayor en el 2D-SBR usa implícitamente el concepto de propagación de la bisección por la arista mayor introducido y discutido en las referencias [85, 88] con el fin de mejorar los algoritmos basados en la bisección por la arista mayor. Este concepto es equivalente al de *grafo* asociado a una triangulación como se hizo notar en [68], aunque no se utiliza explícitamente como estructura de datos.

2. El paso de la bisección de la arista asegura la conformidad de la malla

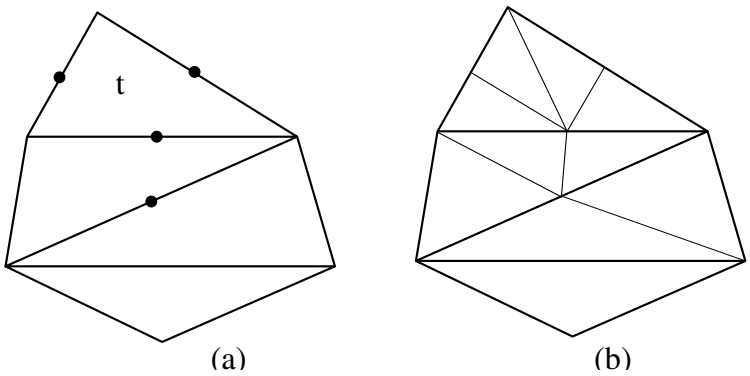


Figura 2.21: Ejemplo del algoritmo de refinamiento 2D basado en el esqueleto para refinar t

refinada. De hecho la división de un triángulo individual se puede realizar en cualquier orden.

3. El algoritmo 2D basado en el esqueleto y el algoritmo 4T, son equivalentes en el sentido de que producen triangulaciones idénticas.

El algoritmo 3D-SBR de Plaza y Carey [68, 69, 71] se puede representar esquemáticamente diciendo que para cada tetraedro t (que va a ser refinado) en cualquier malla conforme de tetraedros τ, hace:

Algoritmo de Refinamiento basado en el esqueleto 3D (τ, t)
Encuentra-y-divide-las-aristas
Encuentra-y-divide-las-caras
Divide-los-tetraedros

Se puede destacar aquí que los procedimientos *Encuentra-y-divide-las-aristas* y *Encuentra-y-divide-las-caras* actúan sobre el *esqueleto* de la malla tridimensional, es decir, sobre la malla de *superficie* definida por las caras triangulares de los tetraedros. Estos procedimientos realizan las siguientes tareas (aunque no en ese orden exactamente): (1) búsqueda tri-dimensional de los tetraedros que participan en el refinamiento, (2) bisección de cada una de las aristas y partición en 4-Triángulos de cada cara del triángulo objetivo, y (3) bisección de cada una de las aristas y partición parcial de las caras que participan en el refinamiento por conformidad de los tetraedros vecinos.

Nótese que, con pequeños cambios los dos primeros procedimientos corresponden a la aplicación al esqueleto de la triangulación inicial del algoritmo 2D-SBR. Por otro lado, el procedimiento Divide-los-tetraedros realiza la partición en volumen del conjunto de tetraedros cuyas caras han sido refinadas previamente. En el capítulo 4 se desarrolla más extensamente una versión del algoritmo 3D-SBR que usa explícitamente la estructura de datos tipo grafo.

2.6. Comportamiento asintótico de las particiones

Seguimos en esta sección las referencias [74, 75]. La pregunta sobre cuál es el máximo número de esferas en tres dimensiones que pueden tocar la esfera unidad, fue causa de una discusión entre Isaac Newton y David Gregory. Newton conjeturó que ese número era 12 mientras que Gregory pensaba que era 13 el número buscado. No fue sino hasta 180 años más tarde cuando Hoppe (1874) demostró que Newton estaba en lo cierto [40].

Para un conjunto convexo K, denotemos por $N(K)$ el máximo número de copias de K que pueden tocar a K sin solapamiento. $N(K)$ se llama *número de Newton* de K [62]. Se conoce el valor exacto del *número de Newton* para la esfera unidad en espacios de dimensión 2 y 3. Obviamente $N(B^2) = 6$, y como ya queda dicho $N(B^3) = 12$. El problema de encontrar el número de Newton para esferas en espacios de dimensión superior es extremadamente difícil, y tiene su propio interés en el campo de la geometría computacional (ver [62], pág. 266 y sg.).

En una malla de símplices, se define el grado de un nodo N como el número de nodos adyacentes a N.

2.6.1. Una medida de irregularidad topológica

Los conceptos de número de Newton y de grado de un nodo se han utilizado últimamente, para definir una medida de irregularidad topológica para mallas de símplices (triángulos en dimensión dos, tetraedros en dimensión tres) [25, 93]. Es fácil observar que una triangulación equilátera tiene en todos sus

nodos interiores grado igual a 6. De ahí que se haya definido como medida de regularidad topológica para triangulaciones en dimensión dos y tres:

$$\epsilon_t = \frac{1}{n}\sum_{i=0}^{n}\Big|\delta_i - D\Big| \tag{2.1}$$

siendo δ_i el grado del nodo i, n el número de nodos de la triangulación, y D el número de Newton para la esfera, en el espacio correspondiente [93].

En esta sección se estudia el comportamiento asintótico del grado de cada nodo, en media, cuando el número de refinamientos globales tiende a infinito. Para este estudio se han usado las funciones de generación ligadas a las relaciones de recurrencia que definen el algoritmo.

2.6.2. Comportamiento asintótico de la media del grado del nodo

Las distintas formas de refinar un triángulo descritas en la sección 2.2 podemos englobarlas dentro de un tipo genérico de particiones que quedan definidas por las siguientes ecuaciones de recurrencia:

$$N_n = dN_{n-1} + eE_{n-1} + fT_{n-1} \tag{2.2}$$

$$E_n = bE_{n-1} + cT_{n-1} \tag{2.3}$$

$$T_n = aT_{n-1} \tag{2.4}$$

donde N_n, E_n y T_n representan el número de nodos, aristas y triángulos respectivamente en cada nivel de refinamiento n, con N_0, E_0 y T_0 dados por la triangulación inicial. Los parámetros a, b, c, d, e, f configuran un tipo u otro de partición, siendo:

a: número de triángulos que se crean por cada triángulo en cada paso.
b: número de aristas que se crean por cada arista existente.
c: número de aristas que se crean en el interior de cada triángulo.
d: número de nodos que se crean por cada nodo.
e: número de nodos que se crean sobre cada arista existente.
f: número de nodos nuevos en el interior de cada triángulo.

Así tenemos que, para el caso de la partición similar 4T y la 4T-LE, los parámetros son: $a = 4, b = 2, c = 3, d = 1, e = 1$ y $f = 0$. Nótese que aunque

las ecuaciones de recurrencia están asociadas a las particiones, no las caracterizan por completo, pues particiones distintas pueden dar lugar a las mismas ecuaciones de recurrencia. Además se puede señalar que en triangulaciones conformes, los parámetros a, b, c, d, e y f no son independientes entre sí. Todas las triangulaciones que aparecen han de cumplir la relación de Euler-Poincaré.

Nos planteamos el siguiente problema: estudiar el límite del promedio del grado de los nodos cuando el número de refinamientos (globales) realizado, n, tiende a infinito para algoritmos basados en la bisección de símplices. Y, más en general, el límite de las relaciones de adyacencia en media, cuando el número de refinamientos tiende a infinito.

Lema 2.6.1 *Sea una triangulación bidimensional con T_n triángulos, N_n vértices y E_n aristas, entonces el promedio de triángulos por vértice viene dado por la fórmula*

$$\mu_n = \frac{3T_n}{N_n} \tag{2.5}$$

y el de aristas por nodo:

$$\mu_n = \frac{2E_n}{N_n}. \tag{2.6}$$

Prueba

Estudiemos primero el caso del promedio de triángulos por nodo. El grado medio de los nodos de la triangulación es $\frac{1}{N_n}\sum_{k=1}^{N_n}\delta_k$, entonces basta con calcular $\sum_{k=1}^{N_n}\delta_k$, y comprobar que este resultado es $3T_n$. En efecto, si realizamos el cómputo por triángulo, el grado de cada vértice se incrementa en una unidad, por tanto la suma del grado de los nodos es $3T_n$. De igual forma se demuestra para el caso de promedio de aristas por nodo, notando en este caso que cada arista posee dos vértices. □

Lema 2.6.2 *Sea una triangulación tridimensional con T_n tetraedros y N_n vértices, entonces el promedio de tetraedros por vértice viene dado por la fórmula*

$$\mu_n = \frac{4T_n}{N_n} \tag{2.7}$$

Prueba
Aplicando el mismo razonamiento de la prueba del lema 2.6.1 se llega a la prueba, notando que en este caso cada tetraedro tiene cuatro vértices. □

2.6.3. Funciones de generación

Una forma de obtener el número de vértices y elementos (triángulos en dimensión dos, tetraedros en dimensión tres) para estas particiones, es hallar primero las funciones de generación asociadas a las relaciones de recurrencia que vienen definidas por las propias particiones. La obtención de estos resultados depende de la existencia de estas relaciones de recurrencia. Sean estas funciones $N(x),\ T(x)$ para los vértices y los elementos respectivamente. Una vez halladas las funciones de generación, es fácil, mediante expansión en serie de potencias hallar las expresiones genéricas para el número de vértices y el número de elementos tras n refinamientos globales: $N_n,\ y\ T_n$, ver [32, 101], pues éstos son respectivamente, los coeficientes de los términos de potencia n-ésima para la x en: $N(x) = \sum_{n\geq 0} N_n x^n,\ y\ T(x) = \sum_{n\geq 0} T_n x^n$, respectivamente.

Teorema 2.6.1 *Sea τ_0 una triangulación bidimensional conforme inicial, en la que efectuamos particiones de los triángulos basadas en las ecuaciones de recurrencia (2.2), (2.3), (2.4). Entonces el límite del promedio de triángulos por nodo tras n refinamientos globales es igual al número de Newton en dimensión 2:*

$$\lim_{n\to\infty} \frac{3T_n}{N_n} = 6$$

Prueba
Algunas relaciones entre los parámetros $a,\ b,\ c,\ d,\ e$ y f son:

1. $d = 1$; lo que es obvio ya que en cada paso de refinamiento no se generan nodos nuevos a partir de los existentes.

2. $e = b-1$; pues si se inserta un nodo en una arista, la arista queda dividida en dos; si se insertan dos nodos, se divide en tres y así sucesivamente. Puesto que el parámetro e indica el número de nodos que se inserta en cada arista existente y el parámetro b el número de aristas que se generan por cada arista antigua, se obtiene la relación apuntada.

3. $f = 1 - a + c$; según la fórmula de Euler: $N - E + T = 1$ aplicada a las ecuaciones de recurrencia para $n = 1$, tenemos:

   $T_1 = aT_0$

   $E_1 = bE_0 + cT_0$

   $N_1 = dN_0 + eE_0 + fT_0$

   Aplicando la fórmula de Euler y las relaciones anteriores, 1 y 2, se llega a la relación $f = 1 - a + c$.

De las relaciones anteriores, las ecuaciones de recurrencia pueden escribirse como sigue:

$$\begin{array}{rcl} N_n & = & N_{n-1} + (b-1)E_{n-1} + (1 + \frac{a}{2} - \frac{3b}{2})T_{n-1} \\ E_n & = & bE_{n-1} + \frac{3(a-b)}{2}T_{n-1} \\ T_n & = & aT_{n-1} \end{array} \tag{2.8}$$

Comenzamos resolviendo la ecuación de recurrencia de los triángulos: $T_n = aT_{n-1}$. Considerando la condición inicial, es decir, que inicialmente tenemos T_0 triángulos, resulta que la función de generación es $T(x) = \frac{T_0}{1-ax}$, o lo que es lo mismo, $T_n = T_0 a^n$.

De la expresión de $T(x)$ y de la ecuación para las aristas se obtiene:

$$E(x) = bxE(x) + \frac{3(a-b)}{2}xT(x) + E_0 \tag{2.9}$$

de donde, sustituyendo $T(x)$ por su valor y despejando $E(x)$, resulta:

$$E(x) = \frac{\frac{3}{2}T_0}{1-ax} + \frac{\frac{3}{2}T_0 + E_0}{1-bx} \tag{2.10}$$

lo que equivale a que los respectivos valores de E_n son:

$$E_n = \frac{3}{2}T_0 a^n + \left(\frac{3}{2}T_0 + E_0\right) b^n \tag{2.11}$$

Finalmente, para hallar la función de generación de los nodos, $N(x)$, hemos de utilizar la primera de las ecuaciones de recurrencia, y las funciones ya calculadas $E(x)$ y $T(x)$, resultando:

$$N(x) = xN(x) + (b-1)xE(x) + \left(1 + \frac{a}{2} - \frac{3b}{2}\right)xT(x) + N_0 \tag{2.12}$$

de donde, sustituyendo $T(x)$ y $E(x)$ por sus expresiones y despejando $N(x)$, resulta después de descomponer en fracciones simples:

$$N(x) = \frac{\frac{T_0}{2}}{1-ax} + \frac{\frac{3}{2}T_0 + E_0}{1-bx} + \frac{-2T_0 - E_0 + N_0}{1-x} \tag{2.13}$$

lo que equivale a que los respectivos valores de N_n son:

$$N_n = \frac{T_0}{2}a^n + \left(\frac{3T_0}{2} + E_0\right)b^n + 3T_0 - 1 \tag{2.14}$$

Con los resultados que se han obtenido, para calcular el promedio de triángulos por nodo, cuando el número de refinamientos tiende a infinito, hacemos:

$$\lim_{n\to\infty} \frac{3T_n}{N_n} = \lim_{n\to\infty} \frac{3a^nT_0}{\frac{T_0}{2}} = 6 \tag{2.15}$$

□

Teorema 2.6.2 *Sea τ_0 una triangulación 3D conforme inicial con N_0 vértices, E_0 aristas, F_0 caras, y T_0 tetraedros. Consideremos la aplicación 8T-LE de forma global. Entonces se produce una secuencia de mallas encajadas $\tau_1,\ \tau_2,\ \ldots,\ \tau_n$. En estas condiciones el número medio de tetraedros μ_n que comparten un mismo vértice en la malla τ_n tiende a 24 cuando el número de refinamientos n tiende a infinito. Además, la media del grado de los nodos o media de aristas por nodo tiende a 14.*

Prueba

El refinamiento global de cada elemento de la malla τ_{n-1} por la partición 8T-LE produce directamente una malla conforme τ_n (ver teorema 2.4.4). Por las propiedades de la partición 8T-LE, los números de vértices, aristas, caras y tetraedros de la malla τ_n, respectivamente N_n, E_n, F_n y T_n, verifican en función de los valores N_{n-1}, E_{n-1}, F_{n-1}, y T_{n-1}, las siguientes relaciones de recurrencia:

$$N_n = N_{n-1} + E_{n-1} \tag{2.16}$$

$$E_n = 2E_{n-1} + 3F_{n-1} + T_{n-1} \tag{2.17}$$

$$F_n = 4F_{n-1} + 8T_{n-1} \tag{2.18}$$

$$T_n = 8T_{n-1} \tag{2.19}$$

con N_0, E_0, F_0 y T_0 dados por la malla inicial.

Vamos a resolver las ecuaciones anteriores siguiendo [32, 101]. De (2.19) se tiene $T_n = T_0 8^n$ con función de generación para los tetraedros $T(x) = \frac{T_0}{1-8x}$.

De la expresión de $T(x)$ y de la ecuación (2.18) tenemos:

$$F(x) = 4xF(x) + \frac{8xT_0}{1-8x} + F_0 \tag{2.20}$$

de donde despejando $F(x)$ se obtiene la función de generación para las caras:

$$F(x) = \frac{(8T_0 - 8F_0)x + F_0}{(1-8x)(1-4x)} \tag{2.21}$$

De forma análoga y utilizando ahora la ecuación (2.17) se obtiene la función de generación para las aristas:

$$E(x) = \frac{(20T_0 - 24F_0 + 32E_0)x^2 + (3F_0 - 12E_0 + T_0)x + E_0}{(1-8x)(1-4x)(1-2x)} \tag{2.22}$$

Por último se puede obtener del mismo modo la función de generación de los nodos $N(x)$, usando (2.16) y (2.22):

$$\begin{aligned} N(x) = {} & \frac{(20T_0 - 24F_0 + 32E_0 - 64N_0)x^3 + (3F_0 - 12E_0 + T_0 + 56N_0)x^2}{(1-8x)(1-4x)(1-2x)(1-x)} + \\ & + \frac{(E_0 - 14N_0)x + N_0}{(1-8x)(1-4x)(1-2x)(1-x)} \end{aligned} \tag{2.23}$$

Finalmente, de las funciones de generación $N(x)$ y $E(x)$, desarrollándolas en fracciones simples, y quedándonos con el término más significativo, obtenemos las siguientes expresiones para el número de tetraedros por nodo y el número de aristas por nodo respectivamente:

$$\lim_{n\to\infty} \frac{4T_n}{N_n} = \lim_{n\to\infty} \frac{4T_0 8^n}{\frac{1}{6}T_0 8^n} = 24 \tag{2.24}$$

$$\lim_{n\to\infty} \frac{2E_n}{N_n} = \lim_{n\to\infty} \frac{\frac{7}{3}T_0 8^n}{\frac{1}{6}T_0 8^n} = 14 \tag{2.25}$$

□

Algunos ejemplos numéricos

En la tabla 2.1 se presenta un ejemplo numérico que muestra que empíricamente $\lim_{n\to\infty} \frac{3T_n}{N_n} = 6$ del teorema 2.6.1 aplicado a una malla refinando globalmente según la partición similar 4T.

Cuadro 2.1: Prueba numérica para la triangulación similar 4T

Iteración	Nodos	Triángulos	$\frac{3T_n}{N_n}$
1	6	4	2
2	15	16	3.200000
3	45	64	4.266667
4	153	256	5.019608
5	561	1024	5.475936
6	2145	4096	5.728671
7	8385	16384	5.861896
8	33153	65536	5.930323
9	131841	262144	5.965003
10	525825	1048576	5.982462

Para ilustrar el comportamiento práctico del algoritmo 8T-LE, se ofrecen los resultados numéricos obtenidos de refinar globalmente la malla compuesta inicialmente por un único tetraedro mediante seis aplicaciones del algoritmo 8T-LE hasta obtener 262144 tetraedros [75]. Se consideran dos casos: el primero correspondiente a un tetraedro casi-regular, y el segundo a un tetraedro casi plano. Las tablas 2.2 y 2.3 ofrecen la evolución de la media del tamaño de las moléculas de los nodos, su desviación típica y el máximo valor del tamaño de la molécula. Por su parte las figuras 2.22 y 2.23 muestran la distribución, en tanto por ciento, del número de vértices por segmentos de números de tetraedros por vértice, de amplitud 5 unidades.

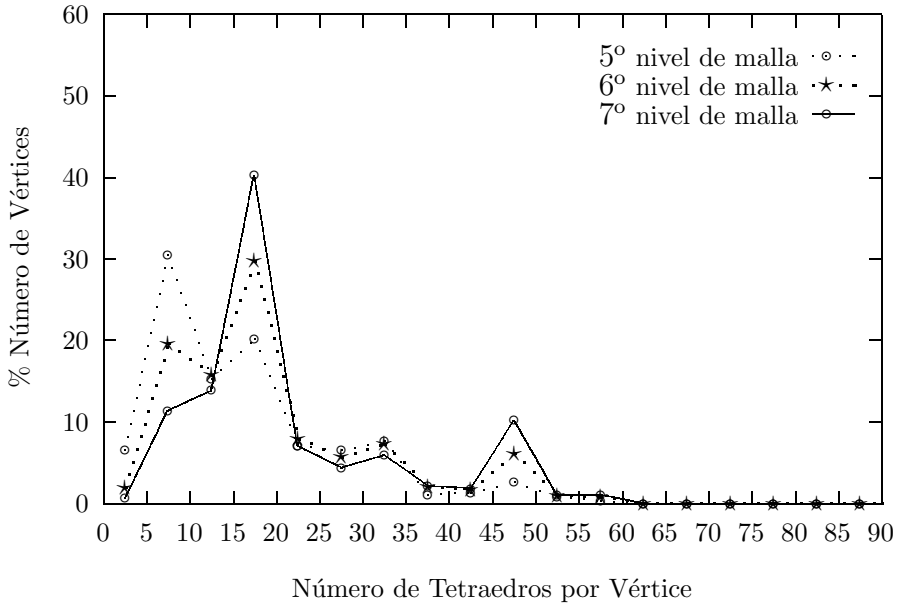


Figura 2.22: Distribución de vértices en % vs. número de tetraedros por vértice. Tetraedro casi-regular

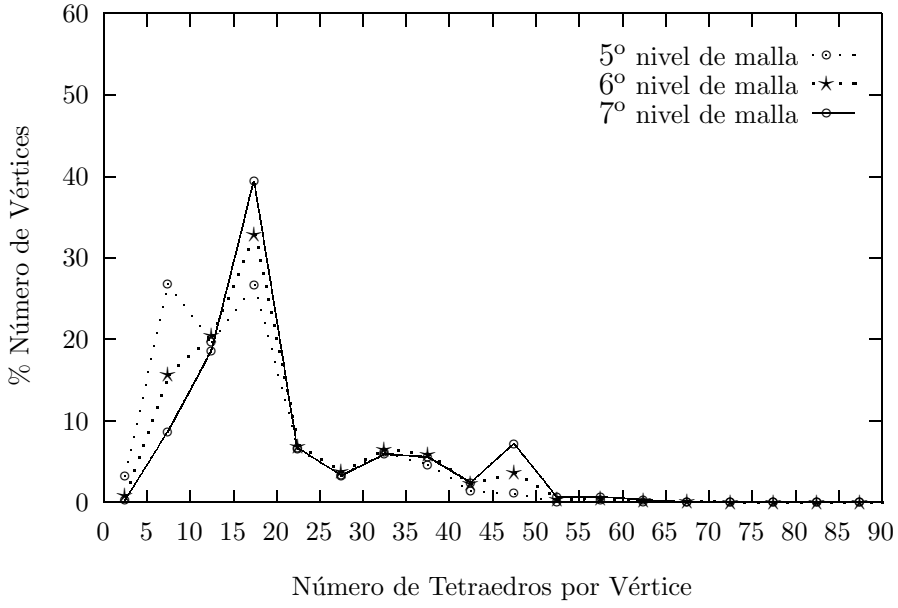


Figura 2.23: Distribución de vértices en % vs. número de tetraedros por vértice. Tetraedro casi plano

Cuadro 2.2: Medidas estadísticas I

Nivel n	$\mu(\lvert M_n\rvert)$	σ_n	Max$\lvert M_n\rvert$
4	12.41	8.79	48
5	16.90	11.03	58
6	20.03	12.21	60
7	21.88	12.81	64

Cuadro 2.3: Medidas estadísticas II

Nivel n	$\mu(\lvert M_n\rvert)$	σ_n	Max$\lvert M_n\rvert$
4	12.41	7.89	48
5	16.90	9.79	68
6	20.03	11.18	90
7	21.88	12.13	108

Capítulo 3

Estructuras de datos geométricas

Las estructuras de datos son decisivas para la eficacia de un algoritmo y de las aplicaciones relacionadas. Esta importancia es aún más notable en algoritmos geométricos debido a la complejidad que conlleva el manejo de las entidades geométricas en 2D y 3D.

En este capítulo se presenta un estudio de las estructuras de datos usadas en la literatura en el ámbito de la generación de mallas, ver por ejemplo la referencia [28]. Hay que notar que cada una de ellas posee su campo de aplicación específico. El conocimiento detallado de cada una de ellas permitirá un completo entendimiento de las necesidades y dificultades a abordar en el diseño de una estructura de datos eficiente para un problema.

3.1. Estructuras de datos en la representación de mallas

En la representación computacional de mallas en 2D y 3D se suele distinguir tres tipos de informaciones:

1. La información geométrica, que describe la información matemática de la malla, como coordenadas de los puntos, descripción de curvas etc.

2. Información de adyacencia, que describe las relaciones topológicas de los elementos que forman la malla. Esta información es necesaria porque a

menudo el algoritmo que trabaja con la malla necesita información de adyacencia entre las primitivas de una malla.

3. Información de atributos, que añade información semántica a los elementos o primitivas de la malla. Por ejemplo, asignar a cada vértice o cara un color, información de altura geográfica para cada nodo, asignar variables de temperatura, presión etc. Esta información de atributos depende del ámbito de aplicación de las mallas.

Decimos que una malla está *completamente definida* si se dispone de informacion suficiente de la geometría, adyacencia y atributos de la malla. Existen formas no completas de definir una malla y sin embargo la información puede ser suficiente para las necesidades de muchos problemas aplicados.

Las estructuras de datos geométricas a menudo se denominan en función de las entidades topológicas o primitivas (vértices, aristas, caras y elementos o polígonos) que la representan. Por ejemplo, una estructura de datos basada en aristas debe su nombre a que implementa la representación de la malla en función de las aristas.

Una primera aproximación al representar una malla 2D es proporcionar la lista de los polígonos en función de los vértices. Otro enfoque simple consiste en representar las aristas de la malla en función de los vértices. En ambos casos se dispone de una lista en la que se representa para cada vértice, las coordenadas geométricas que lo definen. Por ejemplo, la malla de la figura 3.1 (a) se representa, atendiendo a estas dos primeras aproximaciones, según las siguientes listas:

- Lista de polígonos:
  - v_1, v_3, v_2
  - v_2, v_3, v_4
  - v_3, v_4, v_5
  - v_3, v_5, v_6
  - v_3, v_6, v_1

- Lista de aristas:
  - v_1, v_2
  - v_2, v_4
  - v_3, v_4
  - v_4, v_5
  - v_3, v_5
  - v_3, v_1
  - v_1, v_6
  - v_6, v_5
  - v_3, v_2
  - v_6, v_3

Una extensión a 3D de dichas representaciones es inmediata si consideramos los poliedros que forman la malla en lugar de polígonos. En este caso es posible definir una lista que considera el conjunto de las caras en función de los vértices. La ventaja que tienen estas aproximaciones que definen las entidades de una malla en función de los vértices de la misma, es su simplicidad. Si el requerimiento de información es para la visualización de la malla, éstas dos aproximaciones son muy adecuadas, por ejemplo en $VRML$ (Virtual Reality Modelling Languaje) [35], que se está convirtiendo en el estandar para la representación y visualización de objetos 3D así como en $OpenInventor^{TM}$ [98]. Además, la lista de polígonos ha sido la estructura de datos clásica ampliamente utilizada en elementos finitos, y por ejemplo el *Partial Differential Equation Toolbox*, en $Matlab$ [52] usa este esquema en 2D. Sin embargo, estas estructuras de datos no permiten representaciones completas de mallas ya que no disponen de información de adyacencia. Aunque tampoco poseen información de atributos, esto no es un problema ya que es inmediata sin más que asignar a cada vértice, arista o polígono (según el caso) dicha información adicional.

Un siguiente paso a la hora de representar una malla 2D es aumentar la información a almacenar. Consiste en disponer de una primera lista con las coordenadas de vértices y dos listas adicionales, una que representa las aristas en función de los vértices y la otra que contenga las caras en función de las aristas. Una extensión a 3D implica considerar además la lista de los poliedros

en función de las caras. A continuación se representa el tetraedro de la figura 3.1 (b) atendiendo a esta forma descrita:

- Lista de aristas + lista de caras + lista de tetraedros.
  Lista de aristas:

  - $e_1 : v_1, v_2$
  - $e_2 : v_1, v_3$
  - $e_3 : v_1, v_4$
  - $e_4 : v_2, v_3$
  - $e_5 : v_2, v_4$
  - $e_6 : v_3, v_4$

  Lista de caras:

  - $f_1 : e_1, e_2, e_4$
  - $f_2 : e_2, e_3, e_6$
  - $f_3 : e_4, e_5, e_6$
  - $f_4 : e_1, e_3, e_5$

  Lista de tetraedros:

  - $t_1 : f_1, f_2, f_3, f_4$

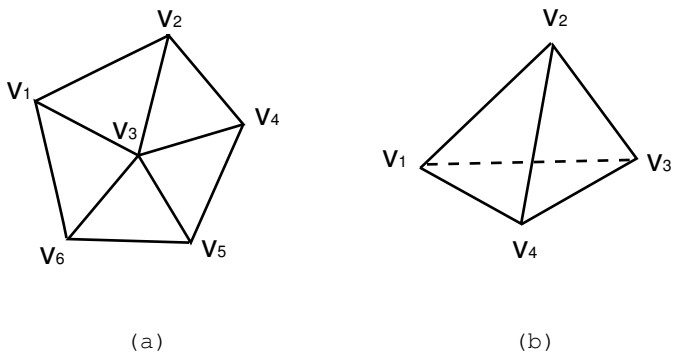


Figura 3.1: Ejemplos de mallas en 2D y 3D

Este esquema es el usado por ejemplo en el código NEPTUNO [21] en 2D y también en los algoritmos 2D/3D SBR [68, 71] y en el algoritmo de desrefinamiento basado en el esqueleto en 3D [63, 72]. La ventaja de esta representación es que almacena una información jerárquica de la malla ya que define las aristas en función de los vértices, las caras en función de las aristas y los poliedros en función de las caras. Esta cualidad es muy adecuada para los algoritmos de refinamiento basados en el esqueleto, pues dispone de forma jerárquica la información de los conjuntos k-esqueleto para una malla τ. Sin embargo, en líneas generales, esta representación supone almacenamiento excesivo y redundante además de no ser una representación completa por la falta de información de adyacencia.

En las siguientes secciones se estudian estructuras de datos para mallas $2D$ y $2\frac{1}{2}D$ que permiten especificaciones completas de mallas al considerar relaciones de adyacencia entre los elementos de la malla, ver [28, 97].

3.1.1. Relaciones de adyacencia completas

Una forma de representar las relaciones de adyacencia entre las primitivas topológicas del esqueleto es proporcionar enlaces de información (punteros, desde el punto de vista computacional) a cada una de las primitivas adyacentes a una dada. Según este esquema es posible derivar estructuras de datos basadas en vértices, aristas o caras. En la figura 3.2 se presentan esquemáticamente las tres estructuras de datos derivadas:

- (a) Basada en vértices
- (b) Basada en aristas
- (c) Basada en caras.

Analizando las tres variantes de la estructura de datos con adyacencia completa, puede notarse que las estructuras de datos basadas en vértices y caras figura 3.2 (a) y (c) no poseen tamaño fijo, ya que el número de punteros depende del número de elementos incidentes al dado. Esta dificultad implica que el coste de almacenamiento es variable y dependiente de la malla en cuestión. Asimismo, el coste para hacer una consulta depende de la complejidad local de la malla. En general sucede que las dos estructuras de datos ocupan mucho

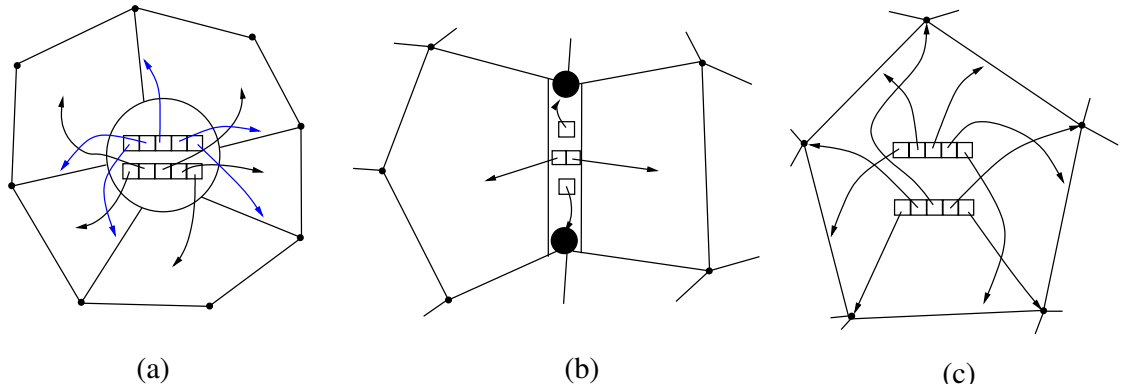


Figura 3.2: Tipos de adyacencia completas

espacio y el coste para mantenerlas en todo momento actualizadas es alto.

Por contra, la estructura de datos basada en aristas, figura 3.2 (b), no posee estos problemas, ya que a lo más, una arista tendrá dos caras y dos vértices adyacentes, salvo cuando pertenece al contorno de una malla, en cuyo caso comparte una. Debido a que la estructura basada en aristas posee esta cualidad la hace eficiente en términos de almacenamiento y se explora en las siguientes secciones, bajo el nombre común de estructura de datos basada en aristas, ya que la información topológica se almacena en relación con las aristas. Para una completa descripción de estructuras de datos basadas en aristas ver [97].

3.1.2. Lista de aristas doblemente conectada (DCEL)

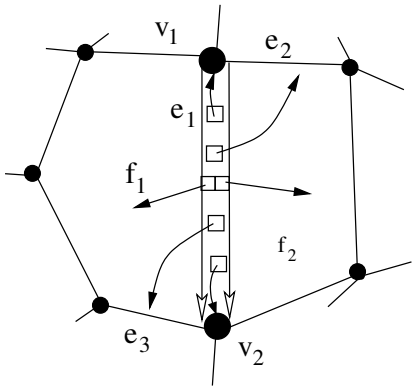
Muller y Preparata [58] diseñaron una estructura de datos denominada *Lista de aristas doblemente enlazada, DCEL*, (Double Connected Edge List). Esta representación trata cada arista teniendo una dirección que impone una orientación a la misma.

Avanzaremos alguna definición relacionada con grafos en esta sección, aunque en el capítulo 4 se realiza un desarrollo más amplio. Un grafo $G = (P, L)$ se dice que está *contenido* en una superficie S cuando se puede dibujar en S tal que dos arcos de L nunca se cortan. Un grafo es *planar* si está contenido en el plano. Un grafo planar puede siempre incluirse en el plano de forma que todos sus arcos sean segmentos de líneas rectas y que además no se corten entre sí. Este tipo de grafos se denomina grafo planar de línea recta, PSLG (Planar

Straight Line Graph) o símplemente grafo planar. Los grafos planares juegan un importante papel en geometría computacional ya que son estructuras que aparecen con cierta frecuencia, por ejemplo en diagramas de Voronoi, triangulación de Delaunay y otras triangulaciones. En lo que sigue se identifican nodos del grafo con nodos de una malla, enlaces del grafo con aristas de la malla, etc.

La DCEL para un grafo planar considera cada arco de L entre (v_a, v_b) conteniendo la siguiente información:

1. V_0, que representa el vértice origen (v_a)
2. V_d, que represente el vértice destino (v_d)
3. F_l, que representa la cara izquierda según la dirección V_0 a V_d
4. F_r, que representa la cara derecha según la dirección V_0 a V_d
5. CCW_0, que representa el sucesor del arco e en el sentido antihorario alrededor de V_0, y
6. CCW_d, que representa el sucesor del arco e en el sentido antihorario alrededor de V_d.



	v_0	v_d	F_l	F_r	CCW_o	CCW_d
e_1	v_1	v_2	f_2	f_1	e_2	e_3
e_2	v_3	v_1	f_2	f_0		
e_3	v_4	v_2	f_1	f_0		
...					...	

Figura 3.3: Diagrama de la DCEL

La figura 3.3 representa un ejemplo de DCEL para el arco e_1 de la subdivisión dada. Según esta estructura de datos, es posible conocer en complejidad lineal las aristas alrededor de una cara dada (en el sentido horario) y las aristas

alrededor de un vértice (en sentido antihorario). Para disponer de esta misma información pero en otro orden específico se deben duplicar los arcos en el grafo planar con orientaciones opuestas a la inicial y de la misma forma construir la DCEL. La complejidad de almacenamiento para esta estructura de datos es $\mathcal{O}(|V| + |F| + |E|)$, siendo $|V|$, $|F|$, $|E|$ el número de vértices, caras y aristas de la malla, respectivamente.

3.1.3. Estructura de datos Winged-Edge

Fue desarrollada originalmente por Baumgart [8] a principios de los 70. La estructura de datos Winged-Edge o Arista con Alas se utilizó para modelar objetos sólidos con superficies poligonales en visión por ordenador. La potencia de esta estructura de datos hace que sea muy adecuada para el modelado de sólidos por dos circunstancias: (1) permite modelar el grafo de las adyacencias topológicas entre caras, aristas y vértices, y (2) paralelamente a la estructura, existe un conjunto de operadores tipo Euler [38] asociados que posibilita la representación del grafo de las adyacencias topológicas sin utilizar detalles inherentes a la estructura de datos.

Dado el grafo planar con vértices y aristas, y los polígonos o caras formando una superficie 2-campo, esta estructura de datos almacena un vector de vértices con la información de la arista adyacente a cada vértice. Asimismo, se dispone de un vector de caras que almacena una arista arbitraria adyacente a la cara. Finalmente, la carga de información se almacena para cada arista $e = (v_a, v_b)$, de ahí que puede considerarse esta estructura de datos como basada en aristas, tomando un vector de aristas que almacena los siguientes punteros, ver figura 3.4 :

1. V_0, que representa el vértice origen (v_a)
2. V_d, que represente el vértice destino (v_d)
3. F_l, que representa la cara izquierda según la dirección V_0 a V_d
4. F_r, que representa la cara derecha según la dirección V_0 a V_d
5. CW_0, la arista siguiente a e alrededor de V_0 en sentido horario
6. CCW_0, la arista siguiente a e alrededor de V_0 en el sentido antihorario

7. CW_d, la arista siguiente a e alrededor de V_d en el sentido horario

8. CCW_d, la arista siguiente a e alrededor de V_d en el sentido antihorario

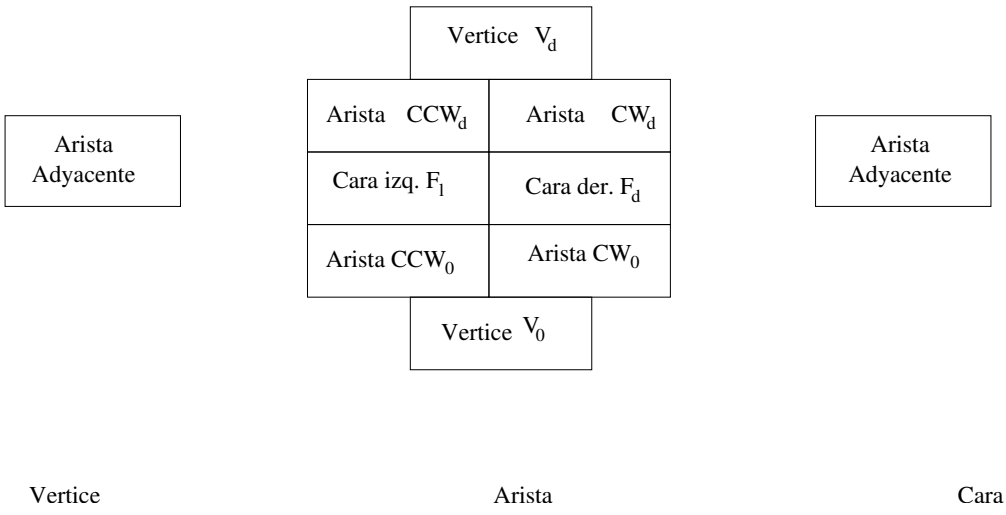
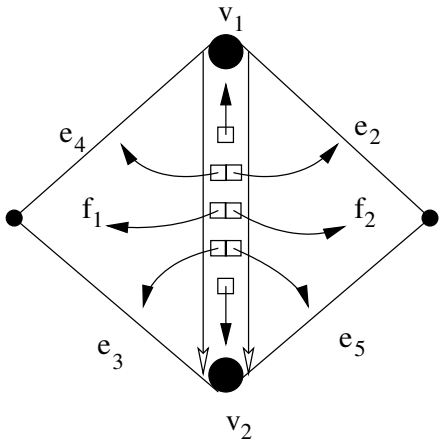


Figura 3.4: Esquema de almacenamiento Winged-Edge



	v_0	v_d	F_l	F_r	CCW_o	CCW_d	CW_0	CW_d
e_1	v_1	v_2	f_2	f_1	e_2	e_3	e_4	e_5
e_2	v_3	v_1	f_2	f_0				
e_3	v_4	v_2	f_1	f_0				
...					...			

Figura 3.5: Ejemplo de almacenamiento Winged-Edge

Para cada arista se almacenan cuatro aristas sucesoras que permiten un acceso eficiente a las aristas alrededor de una cara (sentido antihorario) y de todas las aristas alrededor de un vértice (sentido horario).

Las dos partes simétricas en que se puede clasificar cada arista, lo que el autor denomina *alas*, corresponden a las dos orientaciones posibles para una arista en cuestión. Existe sin embargo un caso ineficiente en esta representación que impide el recorrido de forma concisa de las aristas, y es debido a que el puntero a las aristas sucesoras no tiene información de la orientación concreta que posee la arista. Una extensión en este sentido, codifica en un bit de información la orientación específica.

La estructura de datos Winged-Edge es similar a la estructura de datos DCEL de la sección anterior como se manifiesta en la siguiente proposición.

Proposición 3.1.1 *Sea $e = (v_a, v_b)$ una arista cualquiera de una superficie 2-campo, la estructura de datos Winged-Edge es equivalente a la DCEL eliminando los punteros CW_0, CW_d, la arista siguiente a e en torno a V_0, V_d respectivamente en sentido horario.*

Prueba
Se sigue de forma inmediata sin más que suprimir los punteros CW_0 y CW_d en la tabla de la figura 3.7 y compararla con la tabla de la figura 3.3. □

El coste de almacenamiento para esta estructura es del orden: $\mathcal{O}(|V|+|F|+|E|)$.

A continuación se presenta de forma algorítmica los procedimientos que realizan el recorrido de las aristas en torno a una cara en sentido horario, procedimiento WE_Arista_Cara y el recorrido de las aristas en torno a un vértice en sentido horario, $WE\_Arista\_Vertice$.

Procedimiento WE_Arista_Cara (e_0)
$v = e_0 \rightarrow v_0$
$e = \emptyset$
Mientras $e_0 \neq e$ **hacer**
    Si $v = e \rightarrow v_0$ **entonces**
        $v = e \rightarrow v_d$
        $e = e \rightarrow CCW_d$
    Si no
        $v = e \rightarrow v_0$

$e = e \rightarrow CCW_0$
Fin si
Fin mientras

Procedimiento WE_Arista_Vertice (e_0, v_0)
$v = e_0 \rightarrow v_0$
$e = \emptyset$
Mientras $e_0 \neq e$ **hacer**
Si $v = e \rightarrow v_0$ **entonces**
$e = e \rightarrow CCW_0$
$v = e \rightarrow v_0$
Si no
$v = e \rightarrow v_d$
$e = e \rightarrow CCW_d$
Fin si
Fin mientras

3.1.4. Estructura de datos Half-Edge

La estructura de datos Half-Edge almacena para cada arista en la malla sus dos mitades simétricas. En la figura 3.6 se representa una superficie arbitraria resaltando una primera aproximación a la estructura Half-Edge. En esta figura, cabe notar que para cada mitad de arista se almacenan tres punteros, que son: la mitad de arista simétrica, la mitad de arista siguiente en el sentido anti-horario y el vértice adyacente a la arista. Este primer esquema sin embargo resulta ambiguo ya que no se define explícitamente ninguna orientación de referencia.

Para la estructura de datos Half-Edge, al igual que para las anteriores, es posible asignar una orientación que permite definir de forma única las relaciones con las primitivas adyacentes. De esta forma, asignamos la orientación para cada media arista la que apunta a su vértice adyacente superior, denotado simplemente por v. En la figura 3.7 se representa la misma estructura de datos notando los enlaces de información que se consideran al tener en cuenta la orientación. Puede notarse que para cada mitad de arista se tiene tres tipos de

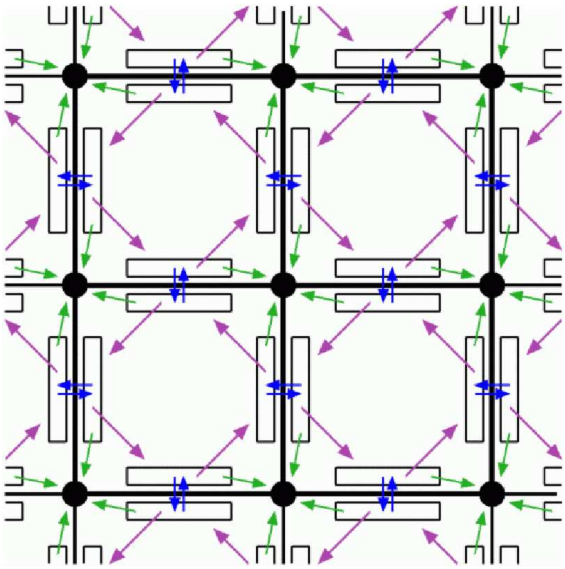
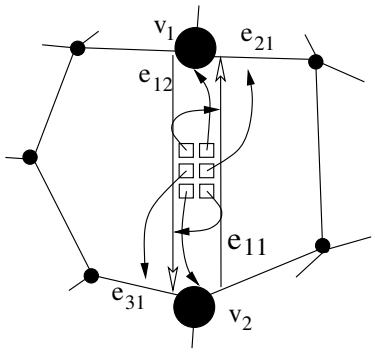


Figura 3.6: Esquema de la estructura de datos Half-Edge

información: (1) el vértice al final de la media arista, (2) arista que es simétrica y opuesta a la actual y (3) arista siguiente en el sentido antihorario. Estas tres informaciones se proporcionan en forma de punteros y en la figura 3.7 se referencian con v, $HEnext$, y CCW. En total se almacenan 6 punteros para cada arista a representar.

Sin embargo, este último esquema se puede completar añadiendo más información a cada arista. Concretamente, para cada media arista se almacena su cara adyacente por la derecha y la media arista predecesora en el sentido horario. Así se llega a la estructura de datos Half-Edge presentada originalmente por Mantyla [55].

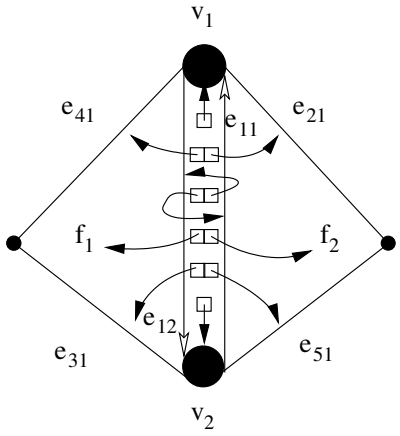
En la figura 3.8 se muestra un ejemplo de este esquema resultante. Se obtiene con este esquema mejorado de la estructura de datos un total de 10 punteros para cada arista a representar. En [44] puede encontrarse una implementación eficiente de dicha estructura de datos para superficies 2-campo.



e

HE	v	HEnext	CCW
e_{11}	v_1	e_{12}	e_{21}
e_{12}	v_2	e_{11}	e_{31}
...			...

Figura 3.7: Ejemplo de representación simple Half-Edge



e

HE	v	F	HEnext	CCW	CW
e_{11}	v_1	f_2	e_{12}	e_{21}	e_{51}
e_{12}	v_2	f_1	e_{11}	e_{31}	e_{41}
...				...	

Figura 3.8: Estructura de datos Half-Edge

Proposición 3.1.2 *En una estructura de datos Half-Edge con punteros v, HEnext y CCW se cumple:*

1. $HE \rightarrow HEnext \rightarrow HEnext = HE$
2. $HE \rightarrow CCW \rightarrow HEnext \rightarrow v = HE \rightarrow v$

□

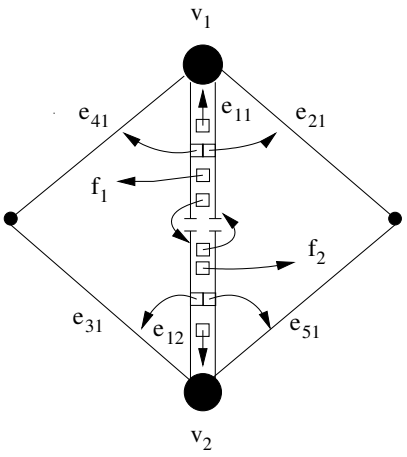
A continuación se presentan dos procedimientos, el primero *HE_Arista_Cara* recorre las medias aristas a partir de una dada HE, en torno a la cara orientada a la derecha de HE y en sentido antihorario. El segundo, $HE\_Arista\_Vertice$ hace el mismo recorrido pero en torno al vértice adyacente a HE.

Procedimiento HE_Arista_Cara (HE)
$Next = HE$
Mientras $Next \neq HE$ **hacer**
$Next = Next \rightarrow CCW$
Fin mientras

Procedimiento HE_Arista_Vertice (HE)
$Next = HE$
Mientras $Next \neq HE$ **hacer**
$Next = Next \rightarrow CCW \rightarrow HEnext$
Fin mientras

Derivada de la filosofía de la estructura de datos Half-Edge surge una posible variante, la denominada Split-Edge. En este caso la arista se divide también en dos mitades que surgen de partir la arista por su centro, quedando así dos partes de aristas adyacentes respectivamente a cada uno de los vértices originales de la arista, figura 3.9. Con esta variante se presentan a continuación los procedientos SE_Arista_Cara y $SE\_Arista\_Vertice$ que realizan respectivamente los recorridos en torno a una cara y a un vértice en sentido antihorario, dada una arista SE.

Procedimiento SE_Arista_Cara (SE)
$Next = SE$
Mientras $Next \neq SE$ **hacer**



e

HE	v	F	HEnext	CCW	CW
e_{11}	v_1	f_2	e_{12}	e_{21}	e_{41}
e_{12}	v_2	f_1	e_{11}	e_{31}	e_{51}
...				...	

Figura 3.9: Estructura de datos Split-Edge

$Next = Next \rightarrow CCW$

Fin mientras

Procedimiento SE_Arista_Vertice (SE)

$Next = SE$

Mientras $Next \neq SE$ **hacer**

$Next = Next \rightarrow CCW \rightarrow SEnext$

Fin mientras

Proposición 3.1.3 *Las estructuras de datos Half-Edge y Split-Edge son duales y se cumple:*

1. $HE\_Arista\_Vertice = SE\_Arista\_Cara$
2. $HE\_Arista\_Cara = SE\_Arista\_Vertice$

□

3.1.5. Estructura de datos Quad-Edge

Guibas y Stolfi [30] desarrollaron una estructura de datos denominada Quad-Edge debido a que cada arista es representada por cuatro mitades simétricas dos a dos.

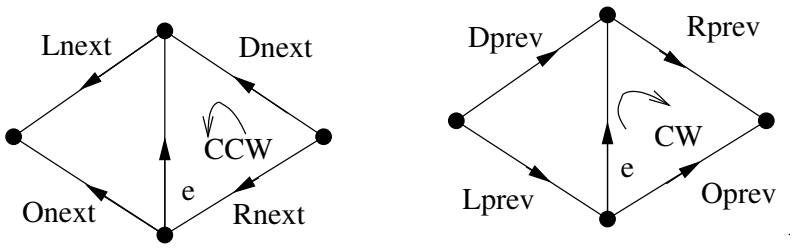


Figura 3.10: Estructura de datos Quad-Edge

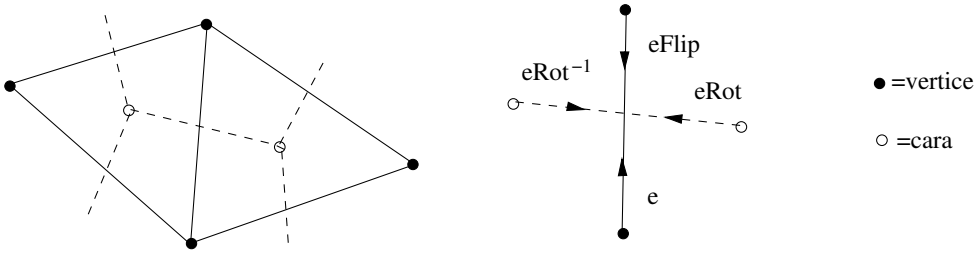


Figura 3.11: Grafo planar y su dual

Dado el grafo planar de una superficie 2-campo, la representación Quad-Edge considera el grafo planar, considerado primal, y su dual simultáneamente, ver figura 3.11. De esta forma se intercambia el concepto de vértice y cara de una malla.

Al igual que las estructuras anteriores, está basada en aristas ya que la información topológica de una malla se almacena en cada arista. Cada arista tiene una orientación y puede modelar mallas o superficies 2-campo orientables y no orientables. En la figura 3.10 se presentan mediante un esquema los punteros de información que se almacenan en la estructura Quad-Edge. Se tienen por cada arista, 8 punteros, cuatro de los cuales equivalen al sentido antihorario, expresado como habitualmente (CCW) y cuatro al sentido horario (CW).

Proposición 3.1.4 *Los punteros Lnext y Rnext permiten realizar el recorrido de aristas adyacentes a las caras izquierda y derecha respectivamente para una arista dada e. Asimismo, los punteros Onext y Dnext permiten realizar el recorrido de aristas adyacentes a un vértice y el opuesto de una arista dada e.*

Prueba

Nótese que dada una arista e, $e \rightarrow Lnext$ proporciona la siguiente arista en el sentido antihorario de la cara izquierda. Si sucesivamente se van obteniendo las aristas $e \rightarrow Lnext$ hasta obtener nuevamente la arista e, se habrá obtenido el recorrido de la aristas que son adyacentes a la cara izquierda. El mismo razonamiento es aplicable para el puntero *Rnext* pero sobre la cara derecha y tomando el sentido horario. Asimismo, para los punteros *Onext* y *Dnext* se sigue el razonamiento anterior, notando en este caso que las aristas que se obtienen son las adyacentes a un vértice y su opuesto. □

Formalmente, la estructura de datos Quad-Edge se define como un álgebra con tres operadores: *Rot*, *Onext* y *Flip*. El operador *Rot* realiza una rotación de 90 grados de la arista, lo que permite oscilar entre la vista primal y dual de la estructura. El operador *Onext* proporciona la arista sucesora en sentido antihorario con respecto a un vértice. Finalmente, el operador *Flip* permite cambiar la vista de la estructura a su opuesta (de arriba a abajo) y viceversa, figura 3.11.

Proposición 3.1.5 *Algunas propiedades de los operadores Rot, Onext y Flip del álgebra que define la estructura de datos Quad-Edge son:*

1. $e\ Rot^4 = e$
2. $e\ Rot^2 \neq e$
3. $e\ Flip^2 = e$
4. $e\ Flip\ Rot\ Flip\ Rot = e$
5. $e\ Rot\ Flip\ Rot\ Flip = e$
6. $e\ Rot\ Onext\ Rot\ Onext = e$
7. $e\ Flip\ Onext\ Flip\ Onext = e$

□

Proposición 3.1.6 *Algunas propiedades del álgebra que define la estructura de datos Quad-Edge son:*

1. $e\ Flip^{-1} = e\ Flip$

2. $e\ Rot^{-1} = e\ Rot^3$
3. $e\ Rot^{-1} = e\ Flip\ Rot\ Flip$
4. $e\ Onext^{-1} = e\ Rot\ Onext\ Rot$
5. $e\ Onext^{-1} = e\ Flip\ Onext\ Flip$

□

Proposición 3.1.7 *Los 8 punteros de la estructura de datos Quad-Edge, figura 3.10 se definen en función de los operadores Rot, Onext y Flip.*

Prueba
Para efectuar la prueba basta con hallar la expresión de los 7 punteros, exceptuando el propio *Onext*, en función de los operadores *Rot*, *Onext* y *Flip*:

1. $e\ Lnext = e\ Rot^{-1}\ Onext\ Rot$
2. $e\ Rnext = e\ Rot\ Onext\ Rot^{-1}$
3. $e\ Dnext = e\ Flip\ Onext\ Flip^{-1}$
4. $e\ Oprev = e\ Onext{-}1 = e\ Rot\ Onext\ Rot$
5. $e\ Lprev = e\ Lprev^{-1} = e\ Onext\ Flip$
6. $e\ Rprev = e\ Rnext^{-1} = e\ Flip\ Onext$
7. $e\ Dprev = e\ Dnext^{-1} = e\ Rot^{-1}\ Onext\ Rot{-}1$

□

La proposición anterior establece que los tres operadores *Rot*, *Onext* y *Flip* son suficientes para calcular las relaciones topológicas de una arista con sus entidades adyacentes en la estructura de datos Quad-Edge. Asimismo, cabe notar que el coste de dichas operaciones es independiente de la complejidad local de la malla.

El coste de almacenamiento para esta estructura es del orden: $\mathcal{O}(|V|+|F|+|E|)$.

3.2. Estructuras de datos espaciales

A diferencia de las estructuras de datos goemétricas estudiadas en la sección anterior, en las que el almacenamiento de la información fundamentalmente es en memoria principal, las estructuras de datos espaciales tienen su mayor interés para el almacenamiento y estructuración en memoria secundaria. Cuando la cantidad de datos geométricos a manejar es de considerable dimensión (varios millones de elementos), el almacenamiento en memoria principal no es eficiente y se recurre al almacenamiento secundario.

Los dispositivos de almacenamiento que generalmente considera un ordenador según el modelo Von Neuman, se dividen en almacenamiento primario (memoria principal o RAM) y almacenamiento secundario (disco). Ambos son dispositivos de acceso aleatorio, es decir, una unidad de información puede ubicarse en cualquier lugar dentro del dominio espacial de la memoria. Los primeros suelen tener capacidades que oscilan en decenas, centenas de megabytes, mientras que los segundos se extienden a varios gigabytes. La memoria principal es más cara que los discos. Pero la diferencia fundamental en términos de estructuras de datos viene definida por los dos parámetros siguientes:

1. Tiempo de acceso. Medido en segundos y representa el tiempo que tarda la CPU de un ordenador en acceder (leer) una unidad de información del dispositivo.

2. Tamaño del bloque de transferencia. Mide la capacidad en bits del bloque de datos unidad que se transfiere desde el dispositivo.

La siguiente tabla muestra medidas típicas de estos dos parámetros en relación a los dispositivos de almacenamiento de memoria principal y disco, tiempo en segundos y tamaño en bits.

En las últimas décadas, el avance tecnológico ha permitido reducir el orden de ambas magnitudes de forma individual. Sin embargo, el ratio entre las dos ha permancecido casi constante. El ratio es importante ya que mide la relación de ambos parámetros al producirse una transacción de información entre un dispositivo y otro o viceversa. Dicha transacción se contabiliza por los accesos a disco y es crucial para las estructuras de datos almacenadas en disco, como

las estructuras de datos espaciales. Además, se consideran en este tipo de estructuras, dos reglas a tener en cuenta en su diseño:

1. Usar una porción de almacenamiento principal, que almacena la estructura de los datos ocupados en disco, y

2. Asegurar que la organización de los datos en disco no degenera cuando se realizan operaciones con los datos de disco, como inserción y eliminación.

Cabe señalar que aunque las estructuras de datos espaciales que se expondrán son orientadas a almacenamiento en memoria secundaria, su uso no queda restringido a éste y es aplicable también a memoria principal, ver [61, 91]. A continuación se describen las estructuras de datos espaciales más importantes.

3.2.1. Acceso Indexado

Como idea general, la estructuración por acceso indexado considera el espacio total dividido en dos zonas. En una de ellas, en disco, se almacena los datos en forma no ordenada. En la otra, memoria principal, se construyen unos índices que permiten acceder rápidamente al bloque en disco donde está el dato que se busca. Esta estructura considera búsqueda por clave única, es decir, a cada unidad de información almacenada se le asocia una clave que la identifica en el almacenamiento. Concretamente, la zona de índices se almacena como un vector relativo en el que existe un registro por cada posible valor de clave, que se supone numérica o traducible a numérica. Cada registro de índice contiene la posición en que se encuentra el registro en la zona de datos, la cual también se maneja como un almcenamiento relativo con los registros ubicados en cualquier sitio libre que hubiese en disco.

Esta técnica resulta muy rápida, pero sólo es apropiada en el caso de que el rango de valores de la clave no sea muy elevado, ya que de lo contrario el índice

Cuadro 3.1: Dispositivos de almacenamiento

Parámetro	Memoria principal	Disco	Ratio
Tiempo de acceso	$10^{-7} \rightarrow 10^{-6}$	$10^{-2} \rightarrow 10^{-1}$	$10^{4} \rightarrow 10^{5}$
Tamaño de transf.	$10 \rightarrow 10^{2}$	$10^{4} \rightarrow 10^{5}$	$10^{2} \rightarrow 10^{3}$

necesitaría demasiado espacio de almacenamiento. La figura 3.12 muestra un esquema del acceso indexado.

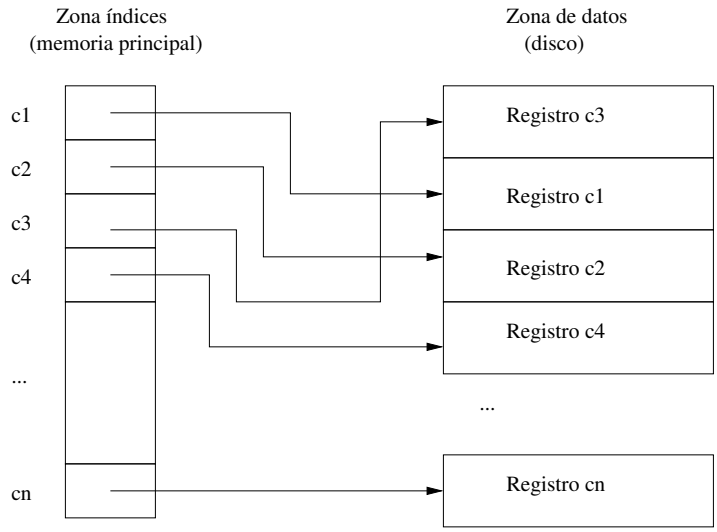


Figura 3.12: Esquema de acceso indexado

3.2.2. Acceso Hash

La dispersión o hashing es una técnica muy utilizada para acceso rápido a información almacenada en dispositivos de memoria secundaria. El planteamiento de dos zonas diferenciadas como en el acceso indexado es análogo. Los registros en memoria secundaria se estructuran por bloques de celdas, enlazados por listas enlazadas. Existe un vector de celdas en memoria principal con B punteros, cada uno indicando la dirección física para cada bloque en disco. Una función de Hash h hace corresponder cada valor de clave con uno de los enteros $[0, B-1]$. Si x es una clave, $h(x)$ es el número celda que apunta a una lista enlazada de bloques donde se encuentra la clave pedida x, si tal registro está presente en disco, ver figura 3.13.

Existen distintas variantes que se puede considerar en el acceso Hash. Primero, si el tamaño del vector de celdas de la memoria principal es excesivo es posible almacenarlo en memoria secundaria y disponer en memoria principal de otro vector de celdas, que en este caso apunta al vector de celdas de disco que a su vez apunta a otra direción de disco en donde se encuentra

el valor pedido con clave x. Así es posible diseñar diferentes niveles de acceso Hash.

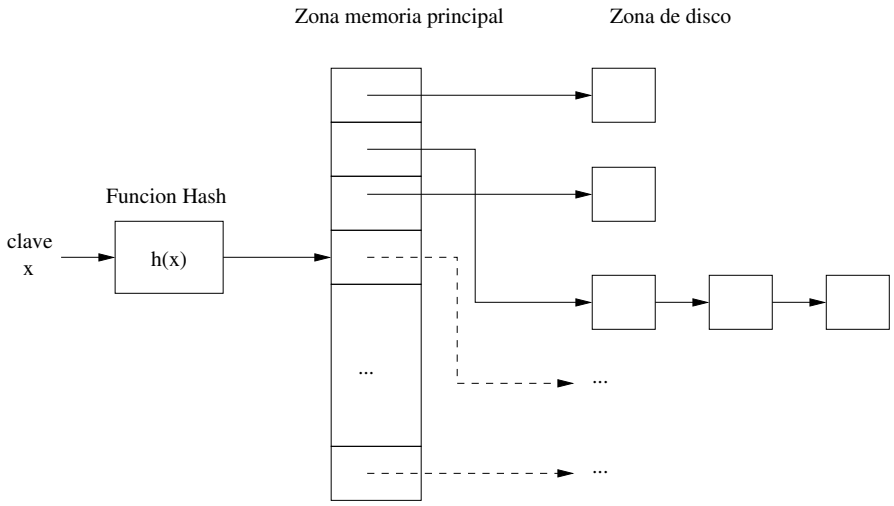


Figura 3.13: Esquema de acceso Hash

Por otra parte, es posible hacer que el tamaño de los registros de disco no sea fijo. Así dependiendo de la aplicación en cuestión se asigna dinámicamente espacio al registro. Un ejemplo posible se encuentra en el siguiente caso. Imagínese que se desea almacenar los triángulos que son adyacentes a un vértice de una triangulación durante un refinamiento global 4T-LE. Una posible aproximación sería asignar como tamaño para 10 triángulos adyacentes a un vértice. Sin embargo del teorema 2.6.1, página 68 es posible conocer que la media del número de triángulos por vértice tiende a 6 cuando el número de refinamientos aumenta. Ello induce a pensar que asignar bloques de tamaño 6 y ampliarlos cuando sea necesario es una idea justificada. Un criterio análogo sería aplicable en 3D, teorema 2.6.2, página 70. En la sección 2.6 se encuentra un estudio del comportamiento asintótico de las particiones del que se derivan resultados aplicables directamente al diseño de estructuras de datos tanto geométricas como espaciales.

Cuando se utiliza este método el problema principal es la elección de la función Hash. Dado que el número de claves posibles puede ser mayor que el número de índices del vector Hash, habrá varias claves que se reflejen en un mismo índice del vector. Cuando surge este problema se dice que hay *colisiones*.

Aunque hay métodos de resolución de colisiones, éstas no son deseables y para evitarlas es necesario que la función que se elija distribuya las claves de forma lo más uniforme posible y por tanto se minimice el número de colisiones. Entre las funciones más usuales se encuentran: funciones con operador *mod*, extracción de bits de la clave, etc.

3.2.3. Integración arbórea

Un *árbol* es una estructura de datos jerárquica de una colección de objetos. Se llama *grado* de un nodo del árbol al número de subárboles (también llamados *hijos*) que tiene. A los nodos con grado cero se les llaman nodos *terminales* o nodos *hojas*. El *nivel* de un nodo es el número de antecesores que tiene desde la raíz del árbol. La $profundidad$ de un árbol se define como el máximo de los niveles de los nodos del árbol. Se llaman *árboles generales* cuando el número de hijos para un nodo es arbitrario. Un caso particular de árboles son los *árboles binarios*, donde cada nodo tiene como máximo grado 2. Los árboles binarios son ampliamente usados en estructuras de datos y existen diversas variantes también de uso extendido. A menudo los nodos de un árbol binario vienen ordenados de forma que, por ejemplo, los hijos por la izquierda de un nodo dado sean menores que el nodo y los hijos de la derecha mayores. La mayoría de las operaciones como búsqueda y adición de elementos a un árbol binario se realizan en $\mathcal{O}(N)$, donde N es el número de nodos en el árbol. Diversos autores han aplicado estructuras de datos basadas en árboles en alguna parte del proceso de generación y/o refinamiento y desrefinamiento de mallas en 2D y 3D, ver [13, 18, 36, 70]. Su naturaleza jerárquica permite que se adapten bien a los procesos también jerárquicos que surgen en la generación y/o refinamiento y desrefinamiento de mallas.

La ampliación multidimensional directa del árbol de búsqueda binario puede orientarse en dos sentidos. El primero de ellos implica la aplicación de la discriminación, en cada nivel, de todas las claves. Entre ellos destacamos el *árbol cuaternario*. La segunda orientación del árbol binario a estructura multidimensional hace que en cada nivel se utilice a una de las claves como discriminante. En esta última categoría destacamos el *árbol kd*. Seguimos acontinuación la referencia [91].

Árbol Cuaternario

Cada registro tiene asociado un punto del espacio bidimensional que se almacena en un nodo de grado cuatro. La raíz del árbol divide el espacio en cuatro cuadrantes llamados: *NE, NO, SO, SE*, por analogía en la orientación espacial geográfica. La raíz de cada subárbol divide a cada uno de los cuadrantes en cuatro subcuadrantes, el proceso se repite recursivamente hasta alcanzar un nodo hoja. Cada nodo consta de cinco campos: un campo $[K_1, K_2]$ que representa el punto bidimensional, y cuatro más que señalan a los cuatro cuadrantes en que se divide el subespacio correspondiente: NE dirección noreste, NO dirección noroeste, SO dirección suroeste y SE dirección sureste. En general, para cualquier nodo se enumeran siguiendo el sentido antihorario. En la figura 3.14 se puede apreciar la correspondencia entre un árbol cuaternario y los puntos asociados en un espacio 2D.

El árbol Cuaternario es una estructura que también puede aplicarse para almacenar la jerarquía en el refinamiento de mallas triangulares 2D. Concretamente, la partición similar 4T y la partición 4T-LE, ver definiciones 2.2.1 y 2.2.3 en las páginas 43 y 46 respectivamente. En ambos casos es preciso asignar una posición de referencia para cada triángulo que permita asignar los campos equivalentes *NE, NO, SO, SE* a los subtriángulos que se generan en cada partición. Para el caso de la partición 4T-LE es suficiente tomar como referencia la posición que deja el lado mayor a la izquierda cuando se gira en sentido antihorario por las aristas.

Un árbol cuaternario se dice que está optimizado cuando todo *nodo* cumple la propiedad de que ningún subárbol del *nodo* puede tener más de la mitad de los nodos que el árbol cuya raíz es nodo. El primer paso para la construcción de este árbol optimizado es ordenar los puntos por la coordenada x y a continuación por la coordenada y. En el árbol resultante la profundidad es del orden de $\mathcal{O}(log_2 N)$.

Árbol Kd

Un tipo de árbol especialmente adecuado para almacenar particiones progresivas de un espacio bidimensional es el *árbol kd*. Los árboles *kd* pueden verse como una implementación eficiente de la generalización multidimensional de

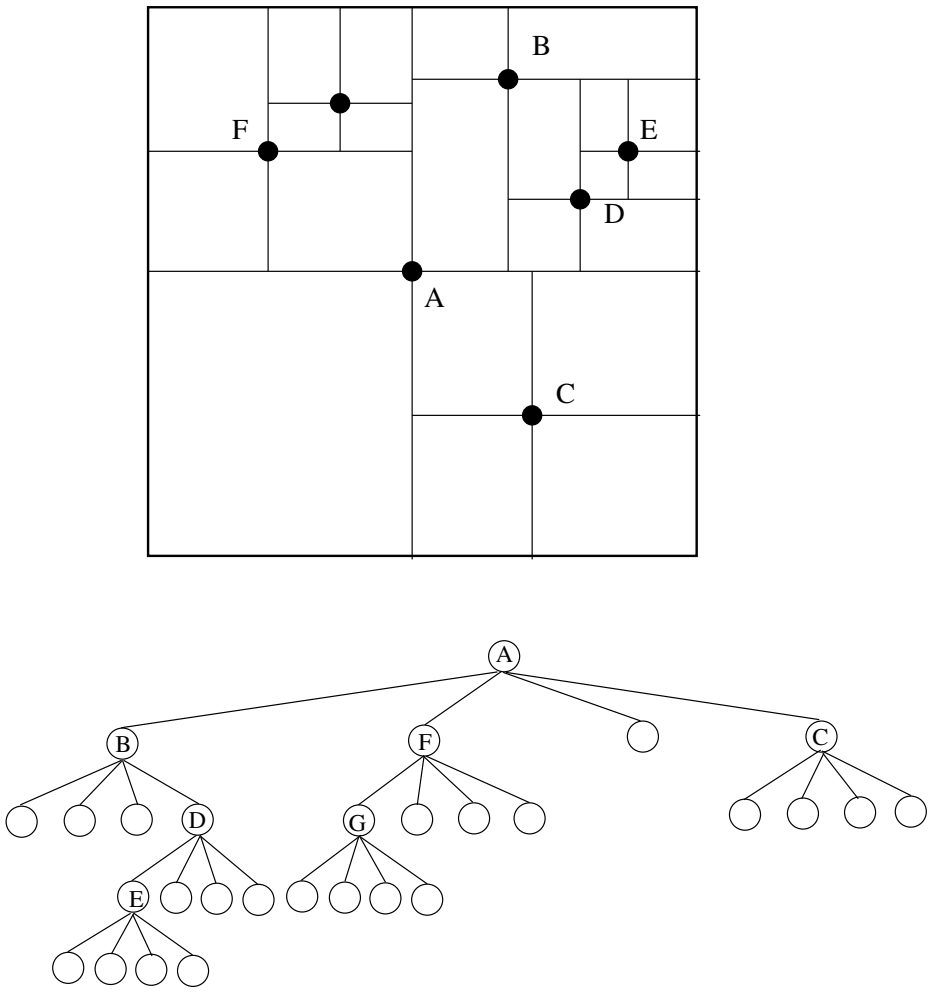


Figura 3.14: Árbol Cuaternario

los árboles binarios y conservan muchas de sus propiedades. En un árbol *kd*, cada registro almacenado en disco se identifica con un nodo del árbol. Cada nodo almacena la siguiente información:

1. Las k-claves $K_0, ..., K_{k-1}$, que definen un punto del espacio asociado al registro.
2. Dos punteros $hizq, hder$ que son nulos o apuntan hacia otro nodo del árbol.
3. Un entero *disc*, perteneciente al intervalo $[0, k-1]$, que indica el subíndice de la clave *discriminante*, y que en adelante se referenciará como discriminante.

La condición que han de cumplir los nodos de cada uno de los subárboles de un *nodo* se puede expresar como sigue:

Si $nodo \rightarrow disc = j$ **entonces**
Para cualquier $nodo^i$ en el subárbol izquierdo de *nodo*:
$$nodo^i \rightarrow K_j < nodo \rightarrow K_j$$
Para cualquier $nodo^d$ en el subárbol derecho de *nodo*:
$$nodo^d \rightarrow K_j > nodo \rightarrow K_j$$
Fin si

Algunas reglas para el dicriminante de cada nodo son:

1. *Todos los nodos de un nivel dado del árbol tienen el mismo discriminante.*
2. *El nodo raíz tiene discriminante 0, sus dos hijos tienen el discriminante 1 y así sucesivamente hasta el nivel k, cuyo discriminante es k-1. El* $(k+1)$*-ésimo nivel tiene, de nuevo, discriminante 0. De esta forma:* $Disc \rightarrow next = (i+1)\ mod\ k$.

En la figura 3.15 se muestra un ejemplo de árbol kd para representar distintos puntos en una espacio bidimensional de dos claves K_0, K_1 que actuan como coordenadas x, y respectivamente. Las operaciones de búsqueda tienen en el árbol *kd* un coste $\mathcal{O}(k \cdot N^{1-\frac{1}{k}})$.

Para el estudio de otras estructuras arbóreas avanzadas como el árbol *bd*, árbol *kdea*, árbol *bm*, árbol *bk* ver [61] y [91].

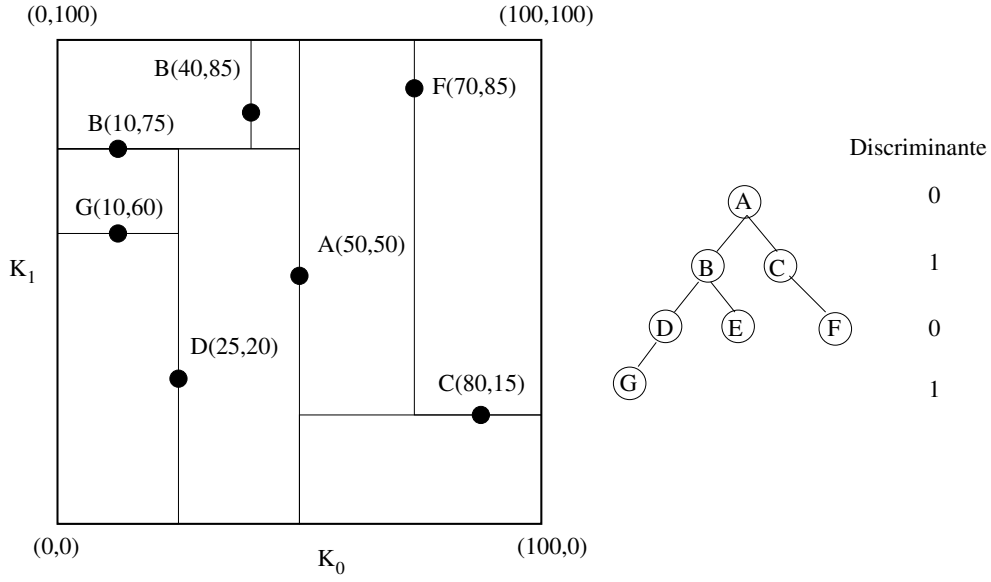


Figura 3.15: Árbol Kd de dos claves

Capítulo 4

Estructuras de datos tipo grafo

4.1. Definiciones y preliminares

La teoría de grafos es particularmente útil para modelar problemas que manifiestan relaciones entre objetos. En este capítulo se presenta el grafo del esqueleto de una triangulación en 2D y 3D, que explícitamente es usado en las estructuras de datos de los algoritmos de refinamiento basados en el esqueleto. Estas estructuras de datos aparecen de forma natural y son consistentes con esta clase de algoritmos.

A continuación se recogen algunos conceptos y definiciones de la teoría de grafos usados más adelante. Estos pueden encontrarse, por ejemplo, en la referencia [29].

Un *grafo geométrico* es un par *(P,L)* donde P es un conjunto no vacío de nodos P y L es un conjunto (posiblemente vacío) de enlaces. El grafo se denota como *G(P,L)* o símplemente *G*. Escribimos $l_k \sim (p_i,p_j)$ para representar el enlace l_k asociado a los nodos (p_i, p_j). Los nodos representan objetos y los enlaces relaciones entre esos objetos. Los enlaces pueden tener *etiquetas* o *pesos* y los nodos pueden tener *nombres*. Un subgrafo $G(P', L')$ es un grafo tal que P' está contenido en P y L' está contenido en L. Un *enlace* $l_k \sim (p_i,p_j)$ entre los nodos p_i y p_j se dice que es *incidente* a los nodos p_i y p_j. Los nodos p_i y p_j se llaman *extremos* del enlace (p_i,p_j). Dos enlaces son *adyacentes* si tienen un nodo común. El *grado* de un nodo p_i es el número de enlaces incidentes a p_i. Un grafo *planar* es un grafo que se puede dibujar en el plano conteniendo todos sus

enlaces de forma que los enlaces sólo se cortan en los nodos del grafo. Un grafo es *conectado o conexo* si todo par de nodos en G están conectados, en otro caso se dice que es *desconectado*. Una *componente* es un subgrafo conectado maximal. Si los enlaces del grafo son ordenados entonces el grafo es *dirigido* y si no lo son se llama *no-dirigido*. Para un grafo dado, si podemos recorrerlo de un nodo a otro por sucesivos enlaces adyacentes sin pasar dos veces por el mismo nodo se llama una *cadena* en un grafo no-dirigido y un *camino* en un grafo dirigido. La *longitud* de un camino o una cadena es igual al número de sus enlaces. Si una cadena o un camino vuelve al nodo inicial se llama *circuito* o *ciclo* respectivamente. La *distancia* entre dos nodos es la longitud del camino más corto que los une. *Recorrer* un grafo consiste en visitar todos sus vértices. El *recorrido en profundidad* (*depth-first traversal*) de un grafo consiste en que, comenzando con un nodo inicial, se visita cada nodo adyacente y cada sucesor de los visitados anteriormente, y así sucesivamente hasta visitar todos los nodos del grafo.

El *camino de propagación por el lado mayor, LEPP, (Longest Edge Propagation Path)* de un triángulo t_0 es definido por Rivara [86] como sigue.

Definición 4.1.1 (Camino de propagación por el lado mayor LEPP) *Sea un triángulo t_0. El camino de propagación por el lado mayor de t_0 es la lista ordenada de todos los triángulos adyacentes $\{t_0,t_1,...,t_n\}$ tal que t_i es el triángulo vecino de t_{i-1} por la arista mayor de t_{i-1}.*

Nótese que al calcular el *LEPP* de un triángulo dado t, t pertenece también a su *LEPP*.

A continuación se define el grafo de incidencia en términos del concepto geométrico de esqueleto de una malla simplicial, el cual nos servirá en la sección 4.2 para definir el grafo basado en el esqueleto como la estructura de datos óptima para los algoritmos de refinamiento y desrefinamiento basados en el esqueleto.

Definición 4.1.2 (Grafo de Incidencia) *El grafo de incidencia $I(P, L)$ de una malla simplicial conforme τ es el grafo no dirigido en el que $P = \{p_0,p_1,...,p_r\}$ es el conjunto de las k-caras, para $-1 < k \leq n$, de τ y los enlaces $L = \{l_0,l_1,...l_s\}$*

cumplen que cada $l_j \in L$ es el enlace que representa la adyacencia entre una k-cara y otra $(k-1)$ o $(k+1)$-cara. Ver figura 4.1.

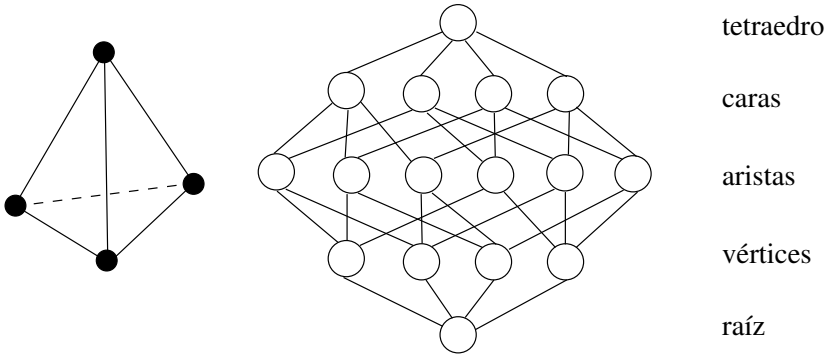


Figura 4.1: Grafo de incidencia para un tetraedro

En el grafo de incidencia se utiliza una (-1)-cara denominada *raíz*. Ver figura 4.1. Esta única cara de dimensión (-1) hace que por ejemplo, el grafo de incidencia de un tetraedro sea simétrico, y que muchas fórmulas que aparecen en el estudio de los complejos simpliciales se simplifiquen.

Nótese que la definición de grafo de incidencia es una representación de una malla simplicial que almacena la estructura de las k-caras de una triangulación τ y las relaciones de adyacencia verticales, es decir, entre una k-cara y las caras del $(k-1)$- y $(k+1)$-esqueleto respetivamente. El grafo de incidencia tiene la restricción de que no almacena información de orden entre los nodos del grafo, lo cual es importante si lo que se pretende es realizar particiones irregulares de la malla, y no almacena las relaciones de adyacencia entre las propias k-caras para un conjunto k-esqueleto dado. La principal ventaja del grafo de incidencia en geometría computacional es que representa el modelo que permite tratar con símplices de dimensión superior a 3 y para este caso se encuentran distintos algoritmos de construcción del grafo en [19].

En la sección siguiente se especifica un nuevo tipo de grafo para una malla simplicial que denominamos *grafo basado en el esqueleto* y que da pie al desarrollo de la estructura de datos que presenta esta tesis.

4.2. El grafo basado en el esqueleto

Partiendo del concepto geométrico de esqueleto de un símplice introducido en la sección 2.1 es posible construir un grafo que representa las adyacencias entre las k-caras de la malla. Por ejemplo, en la figura 4.2 se describe una representación jerárquica de los distintos esqueletos para un tetraedro y que nos permite construir el grafo asociado. En la figura 4.2.(a) se tiene el tetraedro de referencia en 3D, (espacio Euclídeo); (b) muestra las caras una vez abierto el tetraedro a lo largo de las tres aristas que comparten el vértice D y proyectadas en un plano 2D. Como consecuencia resulta una representación del 2-esqueleto del tetraedro. En (c) se representan las aristas del tetraedro original formando el 1-esqueleto. Finalmente la figura 4.2.(d) muestra los cuatro vértices que definen el 0-esqueleto del tetraedro.

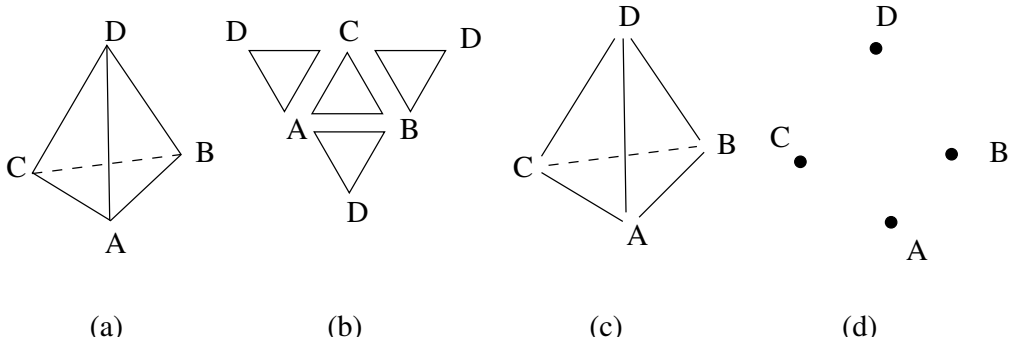


Figura 4.2: Representación gráfica del esqueleto

Definición 4.2.1 (grafo del k-esqueleto $G^k(P, L)$) *Sea τ una malla n-simplicial con el conjunto k-esqueleto, $k < n$. El grafo no-dirigido del k-esqueleto $G^k(P,L)$ de τ se construye como sigue:*

1. *El conjunto de los nodos del grafo $P = \{p_0,p_1,...,p_r\}$ es el conjunto de las k-caras de τ.*

2. *El conjunto de los enlaces $L = \{l_0,l_1,...l_s\}$ es tal que cada $l_j \in L$ es el enlace en el grafo que representa una relación de adyacencia topológica $\mathcal{R}$ entre dos k-caras $\in$ k-esqueleto(τ). Lo denotamos $l_i \sim (p_m, p_n)$.*

Dos nodos diferentes $p_n, p_m \in P$ están relacionados (p_n conectado con p_m) y escribimos $p_n \mathcal{R} p_m$, si y sólo si la k-cara representada por p_n comparte con p_m una *(k-1)*-cara común.

$G^k(P, L)$ no es único si tomamos un conjunto dado k-esqueleto, pues depende de la relación $\mathcal{R}$. Por ello existirá una familia de grafos dependientes de $\mathcal{R}$ para el k-esqueleto de una triangulación τ. Cuando sea necesario identificar cada uno de los grafos de esta familia utilizaremos un subíndice j, G^k_j que los diferencian. Cuando no haga falta tal distinción lo denotamos por $G^k(\tau)$.

Se pueden añadir etiquetas a los enlaces del grafo definido anteriormente. Para cada enlace $l_i \in L$ y nodos $p_n, p_m \in P$ tales que $l_i \sim (p_m, p_n)$ se asigna un número entero que identifica la *(k+1)*-cara o *(k+1)*-símplice en el cual los miembros del k-esqueleto representados por (p_m, p_n) están contenidos. Puesto que dicha etiqueta puede que no sea única, se hace la distinción haciendo que los elementos referenciados superiormente a una $(k+1)$-cara o inferiormente a una $(k-1)$-cara posean una orientación específica. Este es el caso para las aristas por ejemplo, o para las caras. Nótese que esto no supone una restricción para la propia estructura de datos, pues a menudo, los algoritmos de mallas necesitan de esta disintición implícitamente, por ejemplo, en elementos finitos. Asignando cuidadosamente etiquetas a los enlaces del grafo, se puede reconocer la relación superior o inferior de los elementos del k-esqueleto como por ejemplo, $arista \rightarrow cara$, $cara \rightarrow tetraedro$. Esto es muy útil al programar los algoritmos de refinamiento. Además, si importa conocer una relación como por ejemplo $arista \rightarrow tetraedro$, la etiqueta será un único número entero que refiere al tetraedro adecuado de la malla.

La principal diferencia que existe entre el grafo de incidencia y el grafo basado en el esqueleto es que el primero mantiene relaciones de adyacencia verticales entre las k-caras del esqueleto. Sin embargo, el grafo basado en el esqueleto almacena las relaciones de adyacencia horizontales, es decir, para un conjunto dado del k-esqueleto, representa mediante enlaces las relaciones de adyacencia entre las k-caras. Además, el grafo basado en el esqueleto tiene la facultad de almacenar un tipo específico de relaciones verticales, considerando la etiqueta de los enlaces como se explica en el párrafo anterior. Esta cualidad combinada lo hace adecuado en algoritmos que utilicen constantemente las

k-caras para un conjuto k-esqueleto, por ejemplo un algoritmo que divide las aristas de una malla primero, luego las caras y así sucesivamente hasta alcanzar la bisección del símplice. Este es el enfoque precisamente de los algoritmos basados en el esqueleto discutidos e implementados en esta tesis, y por ello el grafo basado en el esqueleto surge de forma natural en la implementación de dichos algoritmos.

Consideremos ahora como ejemplos el triángulo y el tetraedro y sus grafos basados en los 1- y 2- esqueletos respectivamente. En la definición 4.2.1 la especificación del enlace entre los nodos en el grafo está determinada por la relación de adyacencia $\mathcal{R}$ entre las aristas o caras en el n-símplice. En la figura 4.3 se muestran las representaciones de los grafos en los que $\mathcal{R}$ define todas las formas posibles de relación entre aristas y caras tal que el grafo del esqueleto permanezca conectado. Se denota en dicha figura los nodos del grafo sin identificar concretamente el nombre de aristas o caras. Se consideran los patrones de conexión entre los elementos del $(k-1)$-esqueleto, no importando por el momento el orden o especificidad de los nodos.

Para describir precisamente las distintas especificaciones de la relación $\mathcal{R}$ para un grafo $G^k(P, L)$ definimos a continuación el vector de adyacencia de un n-símplice.

Definición 4.2.2 (Vector de adyacencia T^{n-1}) *Llamamos vector de adyacencia T^{n-1}, $n > 1$, de un n-símplice, al vector de enteros de longitud $n+1$ cuyos elementos son los grados de los distintos nodos del grafo no dirigido del $(n-1)$-esqueleto.*

El orden de los elementos en T^{n-1} no es relevante, así que cualquier permutación de T^{n-1} es válida. Por ejemplo, para los grafos de las figuras 4.3.(a.1) y 4.3.(b.4) se pueden usar los vectores T^1=*[2,2,2]* y T^2=*[2,2,3,1]* respectivamente. La importancia de estos vectores es que determinan las conexiones en el grafo del $(n-1)$-esqueleto, y por tanto especifican las relaciones de adyacencia topológica en la malla entre los elementos del $(k-1)$-esqueleto. Por ejemplo, T^1=*[2,2,2]* considera la conexión en el grafo G^1 de una arista con las restantes en un triángulo. Asimismo, T^2=*[3,3,3,3]* representa cada cara de un tetraedro conectado con las restantes caras del tetraedro.

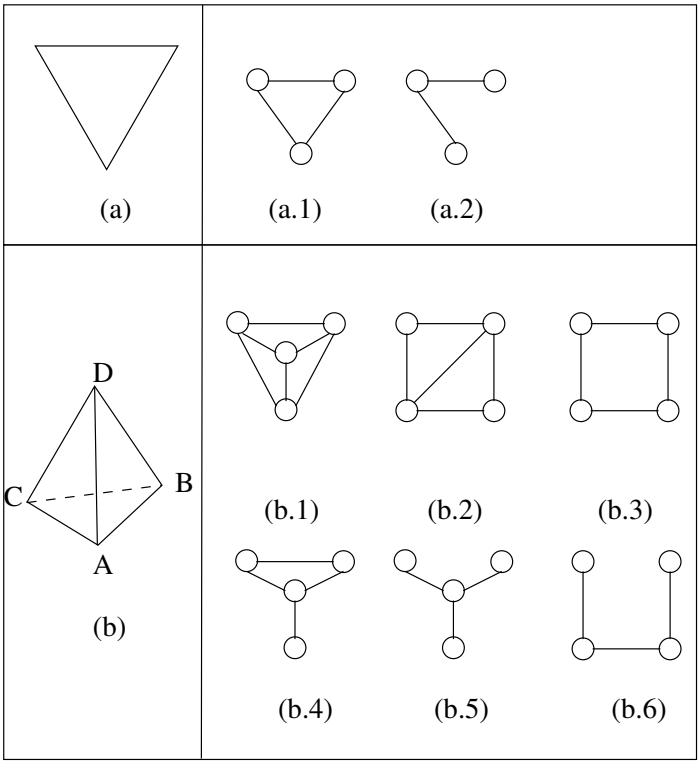


Figura 4.3: Configuraciones posibles para el grafo del $(n-1)$-esqueleto

Un aspecto a tener en cuenta es que la adyacencia definida por T^{n-1} puede restringirse sin pérdida de generalidad mediante, por ejemplo, la minimización del número de relaciones que hay que almacenar para un único n-símplice. Así que, el siguiente funcional es útil para este propósito,

$$T = \min_i \sum_j T_i^{n-1}(j) \tag{4.1}$$

donde el índice i representa los diferentes vectores T^{n-1} para una dimensión dada n (n=2,3) y j indexa cada elemento en el vector.

En la ecuación (4.1), T minimiza el número de enlaces en el grafo del esqueleto para un n-símplice. Según esto, las configuraciones de G^{n-1} para T en dimensiones dos y tres se muestran en la figura 4.4.

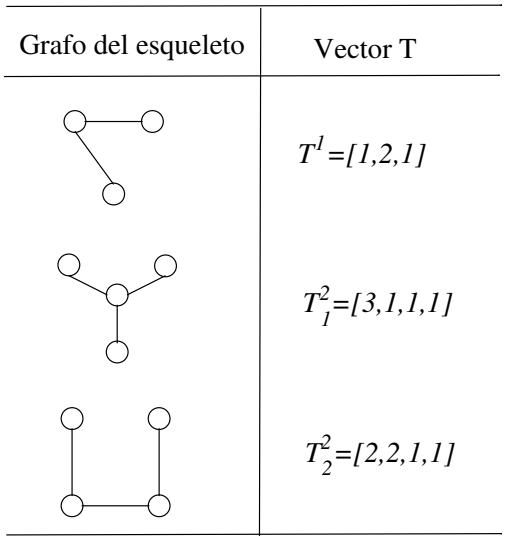


Figura 4.4: Grafo del esqueleto y vector T

Para nuestro objetivo, el uso del grafo en la modelización del esquema de refinamiento basado en la bisección por la arista mayor, restringiremos la relación $\mathcal{R}$ entre las k-caras del grafo del $(n-1)$-esqueleto a aquellas que quedan explícitamente representadas en la figura 4.4. En la próxima sección justificamos esta elección.

4.2.1. El grafo del 1-esqueleto en 2D

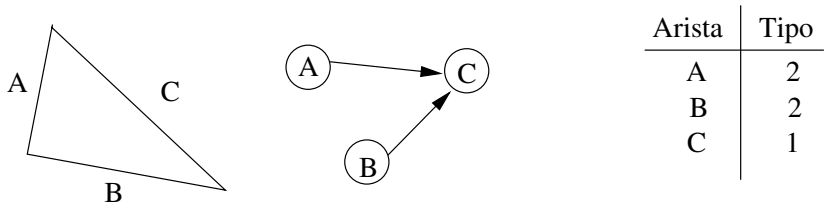
Los algoritmos de refinamiento basados en la bisección por la arista mayor de un triángulo en 2D tienen la ventaja de que las diferentes posibilidades de refinar un triángulo dependen de determinar la arista mayor. En general, se puede decir que estos algoritmos clasifican las aristas de cada triángulo en dos tipos: la arista mayor, de *tipo 1*, y las otras dos aristas, de *tipo 2*, y esto es suficiente para nuestros propósitos (ver figura 4.5) [71].

Esta idea se puede expresar en términos del grafo del 1-esqueleto al considerar la relación $\mathcal{R}$ ="*longitud de arista x en el triángulo es menor que la longitud de arista y*" con el vector T^1 de la figura 4.4. Como $\mathcal{R}$ es una relación de orden entre las aristas, es posible añadir una orientación a los enlaces del grafo que represente este orden. Por tanto, los enlaces en la figura 4.5.(a) indican la dependencia de las aristas al refinar y para asegurar la conformidad. En la figura 4.5.(b) se muestran una sencilla triangulación y el grafo asociado basado en las aristas.

Proposición 4.2.1 *Sea G^1 el grafo dirigido del 1-esqueleto de una triangulación bidimensional conforme τ. Entonces el grafo no contiene ciclos.*

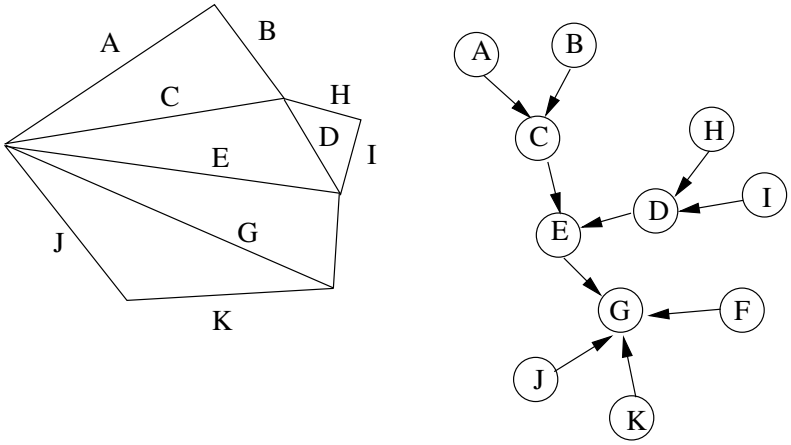
Prueba
Hay dos situaciones en las pueden ocurrir bucles (ciclos). La primera aparece en mallas que tienen triángulos isósceles o equiláteros. En el momento de construir el grafo, en los casos en los que hay triángulos isósceles o equiláteros, se puede dar más de una arista de bisección (arista mayor). En estas situaciones en las que se puede elegir más de una arista mayor, la primera asignada como arista mayor es la elegida. La segunda posibilidad de bucle es la situación descrita en la figura 4.6. Sea X=$\{x_0, x_1, x_2, x_3, \ldots, x_n\}$ las aristas comunes compartidas por triángulos adyacentes en el *LEPP* de t_0 (t_0 es el triángulo definido por aristas c y x_0). Si aparece un ciclo entonces $long(c) < long(x_0) < \ldots < long(x_n) < long(c)$. Sin embargo, esto es imposible ya que las aristas de X son distintas. Si las aristas de X son iguales, estamos en el primer caso considerado. □

Desde el punto de vista computacional la proposición 4.2.2 asegura la localidad de la estructura de datos basada en el grafo dirigido del esqueleto.



Arista	Tipo
A	2
B	2
C	1

(a)



(b)

Figura 4.5: Grafo del 1-esqueleto en 2D

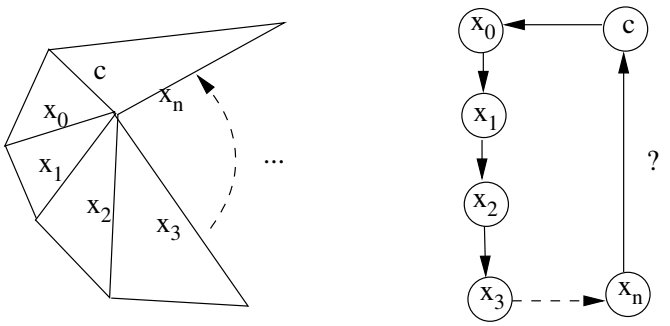


Figura 4.6: Existencia de bucles en el grafo del 1-esqueleto

Proposición 4.2.2 *El coste de actualización de la estructura de datos basada en el grafo dirigido del 1-esqueleto al aplicar la bisección 4T-LE es como mucho tres nodos en cada paso de refinamiento.* □

Proposición 4.2.3 *Sea τ una triangulación conforme bidimensional con N triángulos y t un triángulo arbitrario de τ. Sea e la arista mayor de t. Entonces el LEPP de t se puede obtener de un recorrido en profundidad del grafo dirigido del 1-esqueleto de τ, comenzando por el nodo e del grafo, con un coste máximo del orden de $\mathcal{O}(N-1)$.* □

4.2.2. El grafo del 1- y 2-esqueleto en 3D

Consideraremos primero el grafo del 1-esqueleto de un tetraedro. Como en 2D, se aplica el mismo criterio en cuanto a la relación entre las aristas. En este caso, se pueden distinguir tres tipos de aristas de acuerdo con su longitud (esta clasificación es semejante a la de Bänsch [7], Liu y Joe [48] y Mukherjee [57]). La arista mayor del tetraedro la llamamos arista *tipo 1*. La arista mayor de cada una de las dos caras que no comparten la arista mayor del tetraedro son arista *tipo 2*, y el resto de aristas son aristas *tipo 3* [71].

Observación: Mediante pequeñas (e hipotéticas) perturbaciones de las coordenadas de los nodos puede asegurarse que hay exactamente una única arista mayor en cada tetraedro. La arista mayor de un tetraedro es frecuentemente llamada *arista de referencia.*

Para la partición 8T-LE, distinguimos tres tipos de tetraedros dependiendo de la posición relativa de sus tipos de aristas [71]. Primero consideramos los tetraedros obtenidos al dividir la arista *tipo 1*, después la o las aristas *tipo 2* y así sucesivamente. Cada tipo de tetraedro se puede representar por un grafo orientado del 1-esqueleto como en la figura 4.7. En la figura 4.8 se muestran todas las posibles configuraciones para la partición 8T-LE y los grafos orientados del 1-esqueleto asociados. Nótese que en dicha figura se remarca con línea discontinua la arista mayor de cada cara.

Observación: Algunas propiedades referentes al grafo dirigido del 1-esqueleto para un tetraedro son:

1. El grafo dirigido del 1-esqueleto tiene exactamente 6 nodos y 8 enlaces;
2. El grado del nodo que representa la arista mayor es 4;
3. El grafo dirigido del 1-esqueleto es desconectado y no contiene ciclos.

El grafo dirigido del 1-esqueleto de un tetraedro como ha sido presentado anteriormente es una representación basada en las aristas y se puede utilizar para diseñar la estructura de datos de cualquier esquema de refinamiento de tetraedros basado en bisección por la arista mayor (ver sección 2.2 de la página 43). Esto nos proporciona la información necesaria para llevar acabo el refinamiento de un tetraedro en términos de sus aristas.

Desde el punto de vista computacional, se puede diseñar el algoritmo de refinamiento mediante la construcción local del grafo para cada tetraedro de forma dinámica. Esta es la idea que usamos aquí por eficiencia en la memoria utilizada. De otra forma, se podría construir el grafo y almacenar para cada tetraedro en la malla y entonces actualizar el grafo tras cada refinamiento. Esto implica un almacenamiento extra de datos de $8n$, siendo n el número de tetraedros en la malla.

En la referencia [75] se estudian resultados asintóticos sobre las relaciones de adyacencia entre los elementos topológicos de las mallas en 3D al aplicar la partición 8T-LE. Por ejemplo, cada arista es compartida por 36/7 tetraedros por término medio. Teniendo esto en cuenta, en el grafo del 1-esqueleto, un nodo podría tener en media hasta $(36/7) \cdot 4$ enlaces. Esto implicaría un coste de almacenamiento inaceptable, y por esta razón no es eficiente mantener el grafo del 1-esqueleto de forma global a toda la malla, sino localmente para cada tetraedro.

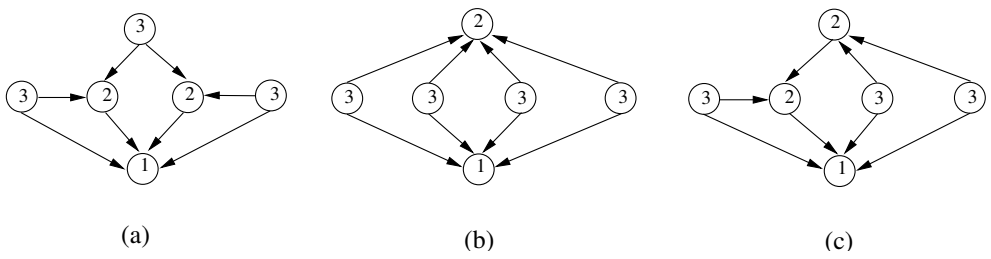


Figura 4.7: Patrones del G^1 para los tres tipos de tetraedros

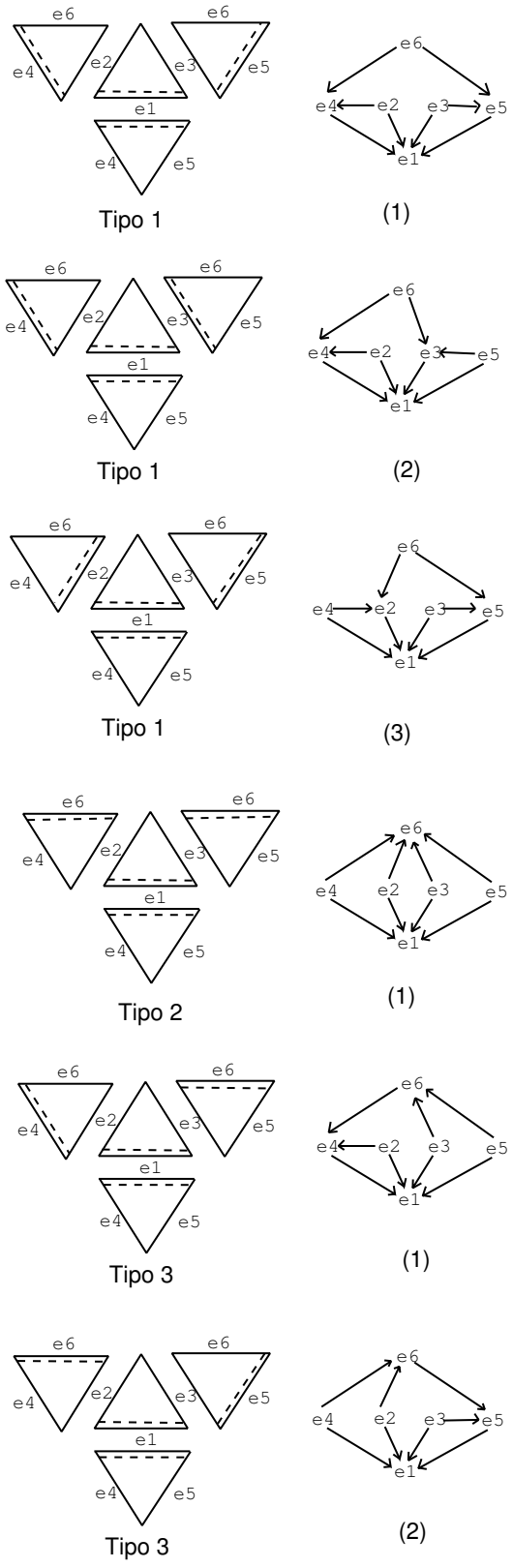


Figura 4.8: Configuraciones posibles de G^1 para los tres tipos de tetraedros

En lo que sigue estudiaremos el grafo del 2-esqueleto como una estructura de datos alternativa para modelar una triangulación 3D.

Observación: La representación de las figuras 4.9 (a)-(b) es única. En este caso la arista AB es la arista mayor. Esta representación se llama *posición estandar*, ver [7]. Llamamos a las caras *f1* y *f4* de la figura, *caras de referencia.*

Tenemos dos alternativas para construir el grafo del 2-esqueleto para un único tetraedro, $G_1^2(\tau)$ y $G_2^2(\tau)$, ver figura 4.9 (c)-(d). Usando la notación del vector de adyacencia presentado en la definición 4.2.2, el grafo $G_1^2(\tau)$ de la figura 4.9.(c) se escribe como T_1^2=*[3,1,1,1]*. En este caso el nodo con grado 3 es una de las dos caras de referencia (si AB es la arista mayor en el tetraedro, entonces las caras de referencia son *f1* y *f4*). La segunda posibilidad respecto del grafo, $G_2^2(\tau)$, se muestra en la figura 4.9.(d) y se denota por T_2^2=*[2,2,1,1]*.

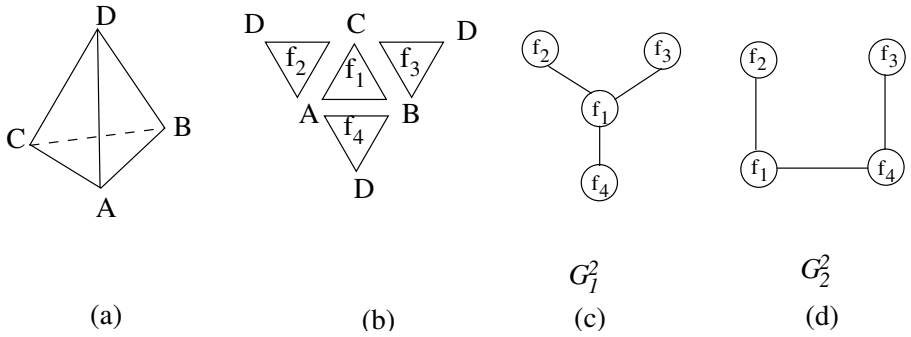


Figura 4.9: 2-esqueleto para un tetraedro

Desde el punto de vista computacional, las configuraciones de los grafos $G_1^2(\tau)$ y $G_2^2(\tau)$ son análogas para nuestros objetivos. Mediante la construcción de dicho grafo, se puede deducir una estructura de datos basada en las caras que es adecuada a las necesidades del algoritmo 3D-SBR como se ve más adelante.

Proposición 4.2.4 *Sean $G_i^2(t_j)$, $i, j = 1, 2$ los distintos grafos no dirigidos del 2-esqueleto de dos tetraedros adyacentes t_1, t_2 respectivamente de una triangulación conforme τ. Sea n el nodo común $\in G_i^2(t_j)$ de las dos componentes de t_j, entonces se cumplen las relaciones siguientes para los grados del nodo común n.*

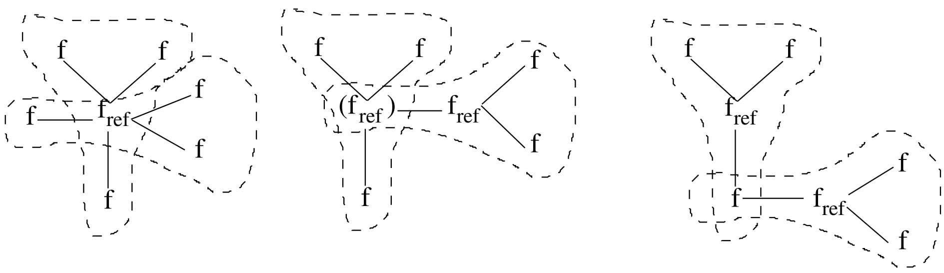
1. *Para $G_1^2(t_i)$, el máximo y el mínimo del grado de n es 6 y 4 respectivamente.*

2. *Para $G_2^2(t_i)$, el máximo y el mínimo del grado de n es 4 y 2 respectivamente.*

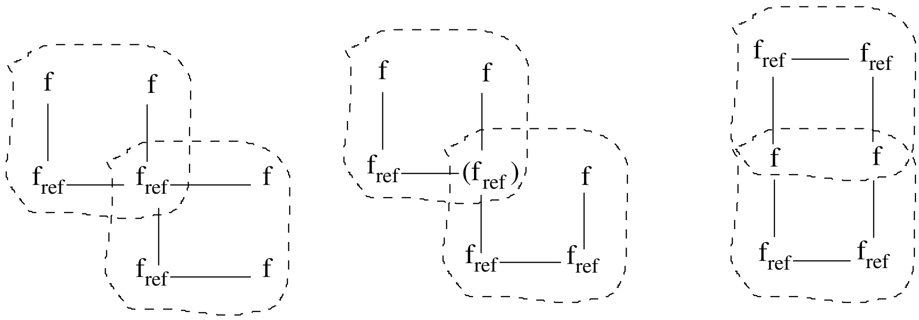
Prueba
Analizaremos las distintas posibilidades de unión entre la cara común de dos tetraedros adyacentes t_1, t_2, atendiendo a los grafos G_1^2 y G_2^2. Un nodo que es común a las dos componentes de los tetraedros adyacentes t_1, t_2 puede ser una cara de referencia (f_{ref}) para algún tetraedro, para los dos o para ninguno. Considerando G_1^2, ver figura 4.9.(c), si el nodo en común es la cara de referencia para ambos tetraedros, ocurre que el grado de este nodo es 6, y es máximo ya que no hay más enlaces posibles para este nodo común que haga tener un grado mayor, ver figura 4.10 (a) en la que se muestran todas las posibles situaciones en este caso. Si consideramos G_2^2, ver figura 4.9.(d), el grado máximo es 4, ver figura 4.10.(b). Si el nodo en común no es cara de referencia para ningún tetraedro, el grado mínimo de este nodo es 2 para ambos grafos y además es el menor posible para dicho nodo común, ver figura 4.10. □

Proposición 4.2.5 *Sea τ una triangulación 3D con n tetraedros y sean, $G_i^2(t_j)$, $i = 1, 2$ y $j = 1, \ldots, n$, los distintos grafos no dirigidos de los tetraedros de τ. Entonces los grafos son planares y conectados.*

Prueba
Primero veamos que son planares. Recordando la definición de grafo planar, un grafo es planar si se puede dibujar en el plano conteniendo todos sus enlaces de forma que los enlaces sólo se cortan en los nodos del grafo. Una componente de los grafos $G_i^2(t_j)$ correspondiente a un tetraedro es planar por definición, ver figura 4.9. Como los grafos $G_i^2(t_j)$ están basados en las caras de la triangulación, la conexión de dos tetraedros es por medio de una cara en común y además cada cara es compartida como mucho por dos tetraedros. Las distintas posibilidades de conexión entre dos componentes según la proposición 4.2.4 son planares y aparecen en la figura 4.10. Entonces, para el grafo completo de una triangulación en la que la conexión de componentes es dos a dos también es planar. Para probar que los grafos son conectados, nótese que cada componente es conectada ya que una cara es adyacente a dos tetraedros. La



(a) G_1^2



(b) G_2^2

Figura 4.10: Componentes con nodo común de G_1^2 y G_2^2

conexión sucesiva de componentes conectadas también origina un grafo que es conectado y esto termina la demostración. □

4.3. Implementación del grafo basado en el esqueleto

El grafo basado en el esqueleto presentado en la sección anterior admite diversas implementaciones, las cuales son comunes a la forma general de representación de grafos, ver [1].

Para representar un grafo dirigido se pueden emplear varias estructuras de datos. La selección apropiada depende de las operaciones que se aplicarán a los nodos y enlaces del grafo. Una representación común para un grafo dirigido $G(P, L)$ es la *matriz de adyacencia*. La matriz de adyacencia para un grafo G es una matriz A de dimensión $n \cdot n$ de elementos booleanos (uno o cero), donde $A[i, j] = 1$ si y sólo si existe un enlace del nodo p_i al nodo p_j. Según esta forma

de representar un grafo, el tiempo de acceso a un elemento es independiente del tamaño de P y A, y por ello, independiente también de la malla.

La principal desventaja de usar la matriz de adyacencia para representar un grafo dirigido es que requiere un coste de almacenamiento $\mathcal{O}(n^2)$, incluso si el número de enlaces del grafo es menor que n^2. Sólo la operación de leer o examinar la matriz A puede llevar un tiempo $\mathcal{O}(n^2)$, lo cual es inaceptable para algoritmos que sean $\mathcal{O}(n)$, como por ejemplo los de refinamiento y desrefinamiento basados en el esqueleto en 2D y 3D. En particular, el grafo G^1 basado en aristas que es la estructura de datos propuesta para los algoritmos de refinamiento y desrefinamiento basados en la bisección en 2D, tiene la característica de que los enlaces indican direcciones del refinamiento en término de las aristas, siguiendo el camino $LEPP$. Un estudio que dejamos para la sección 4.6 de la página 133, demuestra que el número de enlaces recorridos al refinar o desrefinar para una triangulación no degenerada, o después de aplicar sucesivos niveles de refinamiento global a una malla es fijo e independiente del número de nodos. Este dato implica que no son muchos los enlaces entre nodos a almacenar para el grafo G^1 del esqueleto, y que por ello la representación con matriz de adyacencia en sí misma no es eficiente.

Una mejora notable en la forma de almacenar la matriz de adyacencia es almacenarla como *matriz dispersa* (*matriz sparse*). Una *matriz sparse* es una matriz en la que sus elementos son, en la mayoría, ceros. Dichas matrices son muy comunes en simulación numérica y surgen cuando la mayoría de las conexiones, en término de grafos, entre nodos son de caracter local y entre nodos vecinos. Ejemplos de estas matrices son las matrices banda, las matrices diagonales, las matrices triangulares, o simplemente aquellas matrices en la que la mayoría de elementos son ceros.

Una matriz sparse se representa por tres vectores de tamaño q, el número de elementos distintos de ceros, a saber: (1) un vector que almacena los distintos valores distintos de cero, (2) un vector que almacena el índice i del elemento distinto de cero en la matriz original y (2) el índice j del elemento distinto de cero. Con este nuevo esquema de almacenamiento, una matriz sparse requiere $\mathcal{O}(q)$ de almacenamiento, frente a $\mathcal{O}(n^2)$ con el almacenamiento convencional. Sin embargo, debido a que el acceso en una matriz sparse no es directo, ya

que hay que buscar el elemento accediendo a los tres vectores, su uso puede degenerar en tiempo, si no compensa el ahorro de espacio, por ejemplo, si se opta por recurrir a un almacenamiento sparse de una matriz con pocos elementos distintos de cero. Una mejora en el almacenamiento de la matriz sparse se consigue si se almacena siguiendo acceso Hash, ver apartado 3.2.2. De esta forma, el valor de clave es el par (i, j) que referencia al elemento que se desea acceder y el resultado del acceso Hash el valor buscado. La mejora implica un coste de acceso $\mathcal{O}(1)$ e independiente de q, ver apartado 3.2.2.

Finalmente, otra implementación posible de grafos dirigidos es la lista de adyacencia [1]. Se trata de una lista de los nodos del grafo y para cada nodo de esta lista se enlazan en otra lista los nodos que son adyacentes a éste. Esta representación se indica para grafos dirigidos con pocos enlaces. En [70] se presenta una variante de la lista de adyacencia, denominada lista de adyacencia modificada, diseñada específicamente para modelar el refinamiento basado en la bisección por el lado mayor.

4.4. Algoritmos 2D-SBR y 3D-SBR

En esta sección se presentan los algoritmos de refinamiento basados en el esqueleto en 2D y 3D teniendo en cuenta la estructura de datos tipo grafo introducida previamente.

Las particiones usadas al refinar la triangulación en nuestros algoritmos son la 4T-LE para el caso 2D y la 8T-LE para 3D. Con el grafo basado en el esqueleto los algoritmos admiten una descripción natural y consistente.

El algoritmo de refinamiento 2D-SBR

El algoritmo 2D-SBR básicamente realiza el refinamiento mediante dos pasos secuenciales: identifica y divide las aristas de los triángulos a refinar (ver pasos 1 y 2, 2D-SBR más abajo), y subdivide los triángulos individuales para definir una nueva triangulación, (paso 3 abajo).

Algoritmo 2D-SBR (τ, t_0)
/* Entrada: malla τ y triángulo a refinar t_0
/* Salida: malla τ
1: $L = 1 - esqueleto(t_0)$
Para cada arista $e_i \in L$ **hacer**
    Subdivision(e_i)
Fin Para
2: Para cada arista $e_i \in L$ **hacer**
    $S_i = LEPP(e_i, G^1(\tau))$
    Para cada triángulo $t_i \in S_i$ **hacer**
        Sea e_i la arista mayor de t_i
        Subdivision(e_i)
    Fin para
3: Para cada triángulo $t_j \in \tau$ a subdividir **hacer**
    Subsivision(t_j)
Fin para
Fin algoritmo

Subdivisión es un procedimiento que subdivide o bien una arista o bien un triángulo. La subdivisión de una arista consiste en la bisección de la arista por el punto medio. La subdivisión de un triángulo depende del número de aristas divididas. Si tiene sólo una arista dividida, entonces el triángulo es dividido por la arista mayor. Si tiene dos aristas divididas, primero se realiza la división por la arista mayor, y después se unen los puntos medios de las dos aristas divididas. Por último, si tiene las tres aristas divididas, se procede como en el caso anterior y se añade una nueva conexión entre el punto medio de la arista y el punto medio de la tercera arista dividida.

El algoritmo 2D-SBR tiene complejidad lineal con respecto al número de nodos [71]. El algoritmo posee dos puntos en los cuales se centra el coste de cómputo. El primero proviene de la creación y actualización de datos y en este caso, el coste depende de la malla inicial. Las actualizaciones de datos siguientes son locales a cada triángulo como establece la proposición 4.2.2. El segundo punto corresponde al paso 2 para la identificación y bisección de las aristas que intervienen en el proceso de refinamiento. El caso peor ocurre si la triangulación inicial es tal que cada triángulo en el camino de propagación

tiene su arista mayor, mayor que el anterior. Sin embargo, una demostración matemática y experimentos numéricos que se ofrecen en la siguiente sección prueban que el coste en refinar un elemento cada vez es menor y tiende a una constante pequeña a medida que la malla es refinada.

En cuanto al coste de almacenamiento, este es $\mathcal{O}(|V| + |F|)$, siendo $|V|$ y $|F|$ el número de nodos y triángulos en la malla. Se resalta en este punto el ahorro en almacenamiento que supone la nueva versión del algoritmo 2D-SBR aqui presentada, ya que en la versión presentada en [71], el coste de almacenamiento es $\mathcal{O}(|V|+|F|+|E|)$, con $|E|$ el número de aristas totales de la malla.

El algoritmo de refinamiento desarrollado en esta tesis y presentado en [94, 95] aunque es equivalente al de [71] en el sentido de que ante una entrada específica produce la misma salida al final, tiene diversas características que lo diferencian de las versiones anteriores. A continuación presentamos la siguiente comparativa entre los dos algoritmos:

1. La estructura de datos para almacenar la malla pasa de ser (lista de nodos + lista de aristas en función de los nodos + lista de triángulos en función de las aristas) a esta otra, considerada como óptima, (lista de nodos + lista de triángulos en función de nodos).

2. La estructura de datos usada para implementar la partición 4T-LE en el algoritmo está basada en el grafo G^1.

3. Se reducen las líneas de código. La nueva versión implementada en Matlab reduce considerablemente el número de líneas de código de la versión de Plaza y Carey, la cual es código Fortran.

4. La simplicidad de la nueva versión queda manifiesta, aparte de por el reducido número de líneas de código, por la flexibilidad de introducirlo en diversas aplicaciones. Esta característica es en parte debida a la modularidad que ofrece el lenguaje Matlab. Ver capítulo de aplicaciones, donde se usa en un código de elementos finitos comercial y en el lenguaje de realidad virtual $VRML$.

5. La portabilidad del nuevo algoritmo se garantiza debido a las posibilidades de traducción de código Matlab a por ejemplo, lenguaje C, Fortran o Java.

6. La complejidad lineal del algoritmo en términos de tiempo permanece, sin embargo, la complejidad en cuanto al almacenamiento se reduce, ya que no se almacena la lista de aristas de la malla.

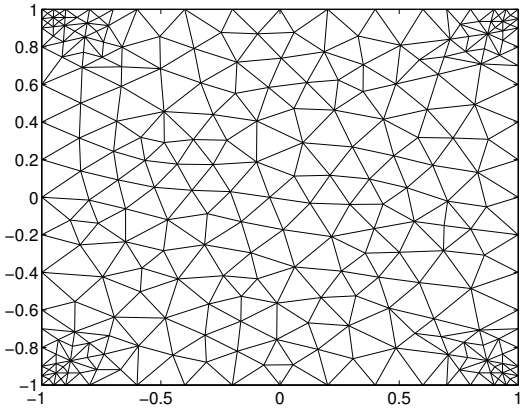
Para ilustrar el uso del algoritmo 2D-SBR, se muestra a continuación un ejemplo test. En las figuras 4.11 y 4.12 se presentan los resultados de aplicación de refinamiento local sobre una malla 2D y la matriz sparse que surge de los conjuntos del 1-esqueleto al refinar la malla en cuestión. En este caso hemos optado por el uso de la partición bisección por el lado mayor. Considerando esta partición, es necesario modificar el paso 1 del algoritmo 2D-SBR como sigue:

1: L=arista_mayor(t_0)
Subdivision(L)

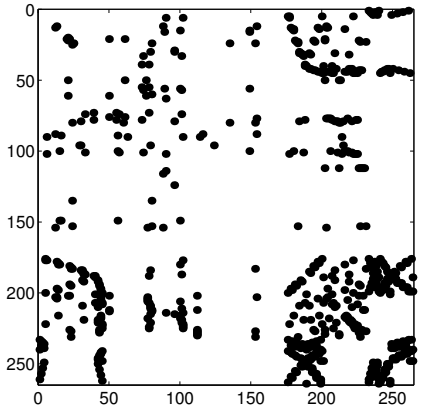
En la tabla 4.1 se presenta de forma numérica los datos de la malla en cada nivel de refinamiento. La matriz sparse que se adjunta para cada nivel tiene orden $p \cdot p$, de forma que la arista (p_i, p_j) es uno, si la arista en cuestión se refina en el nivel de refinamiento realizado. La matriz sparse, aunque puede verse que tiene la apariencia de ser simétrica, nótese que no lo es, ya que las aristas del contorno interiores que quedan refinadas por algún triángulo pero no por el adyacente, no aparecen como simétricas en la matriz.

Cuadro 4.1: Datos de la prueba 1

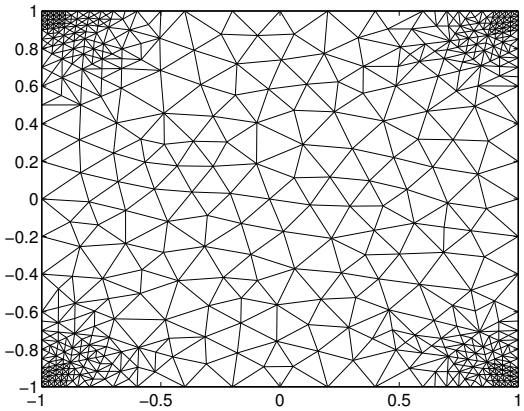
Nivel	Triángulos	Nodos	Aristas	Aristas que se subdividen
1	464	264	727	646
2	1070	587	1656	128
3	1182	651	1832	12



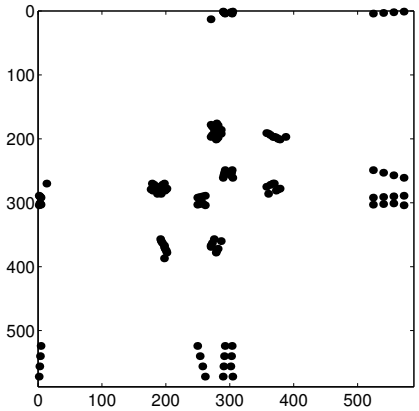
(a) Nivel de refinamiento uno, 464 triángulos



(b) Matriz sparse del 1-esqueleto, 646 aristas

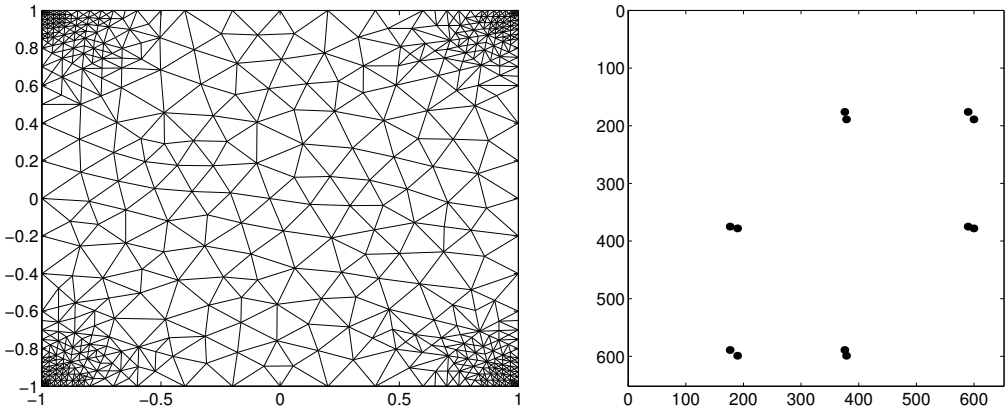


(c) Nivel de refinamiento dos, 1070 triángulos



(d) Matriz sparse del 1-esqueleto, 128 aristas

Figura 4.11: Niveles de refinamiento 1 y 2 del ejemplo test



(a) Nivel de refinamiento tres, 1182 triángulos

(b) Matriz sparse del 1-esqueleto, 12 aristas

Figura 4.12: Nivel de refinamiento 3 del ejemplo test

El algoritmo de refinamiento 3D-SBR

En el caso del algoritmo 3D-SBR, aunque no desarrollamos una implementación del mismo, presentamos una descripción algorítmica con la nueva estructura de datos G^2. Los pasos principales del algoritmo son:

1. Subdivisión de las aristas que forman el 1-esqueleto del tetraedro de entrada.
2. Subdivisión de las aristas debido a la propagación por conformidad.
3. Una vez divididas las aristas implicadas, subdivisión de las caras triangulares correspondientes según la partición 4T-LE.
4. La última etapa realiza la subdivisión del interior de los tetraedros que han tomado parte en el refinamiento mediante la partición 8T-LE.

Presentamos a continuación el algoritmo 3D-SBR en pseudocódigo:

Algoritmo 3D-SBR (τ, t_0)
/* Entrada: malla τ y triángulo a refinar t_0
/* Salida: malla τ
1: $L = 1 - esqueleto(t_0)$
Para cada arista $e_i \in L$ **hacer**
    Subdivision(e_i)
Fin Para
2: Mientras $L \neq \emptyset$ **hacer**
    Sea $e_j \in L$
    Para cada tetraedro $t_i \in hull(e_j)$ **hacer**
        Para cada cara no-conforme $f_i \in G^2(t_i)$ **hacer**
            Sea e_p la arista mayor de f_i
            Subdivision(e_p)
            $L = L \cup e_p$
        Fin para
    Fin para
    $L = L - e_j$
Fin mientras
3: Para cada cara $f_i \in G^2(\tau)$ a subdividir **hacer**
    Subsivision(f_i)
Fin para
4: Para cada tetraedro $t_i \in \tau$ a subdividir **hacer**
    Subsivision(t_i)
Fin para
Fin algoritmo

El procedimiento *subdivisión* subdivide los sucesivos esqueletos en orden *inverso*: pasos 1 y 2 llevan a cabo la subdivisión de las aristas, paso 3 realiza la subdivisión de las caras y el paso 4 subdivide el interior de los tetraedros.

El algoritmo 3D-SBR es de complejidad lineal en el número de nodos [71]. Los puntos donde se usan el grafo del 2-esqueleto $G^2(\tau)$ se observan en el algoritmo anterior: primero, el bucle interior del paso 2 accede a las caras no conformes de la malla y en el paso 3 en la subdivisión del 2-esqueleto. En ambos puntos el grafo del 2-esqueleto proporciona acceso a las caras que toman parte en el refinamiento de forma local y eficiente.

En el paso 2, el procedimiento *hull(e)* proporciona el conjunto de símplices $S \in \tau$ tal que $e \in S$. Es decir, *hull(e)* está compuesto por todos los símplices que comparten la misma arista e. En [71] se da un algoritmo de complejidad lineal para encontrar la envoltura de una arista (*hull*).

Aunque no se desarrolla implementación del algoritmo 3D-SBR en esta tesis, sí puede notarse que la versión aquí presentada del algoritmo minimiza el coste de almacenamiento debido a la elección del grafo G^2 en cualquiera de sus dos opciones, G_1^2 o G_2^2. El grafo almacena la adyacencia de las caras y no es necesario almacenar explícitamente las aristas de la malla, lo que reduce el coste de almacenamiento de $\mathcal{O}(|V|+|E|+|F|+|T|)$ a este otro, $\mathcal{O}(|V|+|F|+|T|)$. En cuanto al coste en tiempo de ejecución, la versión aquí presentada no empeora el caracter lineal, ya que en esencia el esquema algorítmico de refinamiento basado en el esqueleto es el mismo. Se deja como trabajo futuro la implementación del algoritmo, en el que presumiblemente la simplicidad y reducción de líneas de código, sean a buen seguro otras características positivas de la implementación.

4.5. Algoritmo de desrefinamiento 2D-SBD

Tras estudiar los algoritmos de refinamiento tanto en el plano como en el espacio en la sección anterior, y antes de describir con detalle el algoritmo de desrefinamiento 2D-SBD (Skeleton Based Derefinement) que se ha desarrollado, daremos una serie de definiciones relativas a los mallados encajados no estructurados. Aunque nos limitaremos a triángulos, las siguientes definiciones [66, 63] son también fácilmente generalizables a otro tipo de elementos y a mayor número de dimensiones.

Como ya se dijo en la definición 2.1.2 del capítulo dos, dadas dos triangulaciones de un dominio inicial Ω, τ y τ', diremos que están encajadas y que τ es más fina que τ', o que τ' es menos fina que τ, y escribiremos $\tau \geq \tau'$, ó $\tau' \leq \tau$, si se verifican las dos condiciones siguientes:

1. todo nodo de τ' pertenece también a τ.

2. toda arista, cara o tetraedro de τ' se puede poner como unión (conjuntista) de aristas, caras o tetraedros de τ.

Se observa que los respectivos conjuntos de nodos de las triangulaciones verifican la misma relación de contenidos. Es decir, si el conjunto de nodos de la malla τ lo representamos por $N(\tau)$, se verifica:

$$\tau' \subseteq \tau \Longrightarrow N(\tau') \subseteq N(\tau) \tag{4.2}$$

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

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

Definición 4.5.1 (Nodo propio y heredado) *Un nodo* $\mathbf{N}$ *de* τ_j *se dice que es nodo propio de* τ_j *si no pertenece a ningún nivel de malla anterior, es decir, a ninguna* τ_k *con* $k < j$*. En otro caso se dice que el nodo* $\mathbf{N}$ *es heredado en* τ_j*.*

Evidentemente, por la relación de encajamiento entre los conjuntos de nodos, si un nodo $\mathbf{N}$ es propio de τ_j, entonces es heredado en τ_l, $\forall\, l$ con $j < l \leq n$

Definición 4.5.2 (Arista, cara y elementos propios y heredados) *Se definen de forma análoga a los nodos, es decir una arista, cara o elemento -tetraedro-, se dice propio de un determinado nivel de malla, si no pertenece a ningún nivel de malla anterior y heredado en ese nivel si fue creado en algún nivel anterior.*

Definición 4.5.3 (Aristas, caras y elementos padres e hijos) *Se definen de forma natural, por ejemplo, si al refinar, una arista queda dividida en dos, se dice que la primera es la arista padre de estas dos, y éstas se llaman aristas hijas de aquella.*

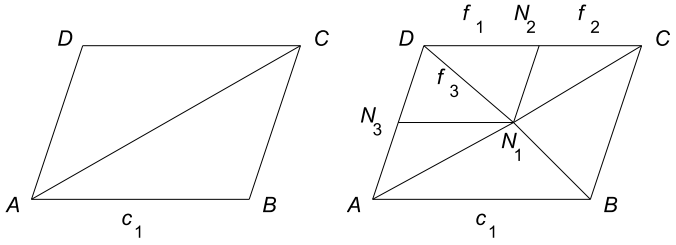
Según el algoritmo de refinamiento que estamos utilizando, una arista solamente podrá tener a lo sumo dos aristas hijas. Hay que notar también que no toda arista tiene arista padre. De forma análoga se definen las caras padres e hijas. Una cara tiene como mucho, cuatro descendientes directos, es decir, cuatro caras hijas, y, como las aristas, no toda cara triangular tiene cara padre. Por otra parte, una cara solamente puede tener cuatro descendientes directos como máximo. Sin embargo, a diferencia de las aristas y de las caras, todo

elemento, excepto los pertenecientes al primer nivel de malla, tienen elemento padre.

Definición 4.5.4 (Aristas internas y externas) *En dimensión dos, al refinar un triángulo $t \in \tau_j$ aparecen nuevas aristas, en el nivel de la malla $j+1$. Pues bien, de estas aristas, las que no tienen arista padre se llaman aristas internas, y las que si tienen las llamamos aristas externas. Las primeras se encuentran en el interior del triángulo t, las demás están en su frontera. En dimensión tres, definimos las aristas internas y externas a un tetraedro aquellas que pertenecen respectivamente a su interior y a su frontera.*

También denominamos arista interna en volumen como aquella que además de no tener padre, no se encuentran en el interior de una cara triangular, sino que es interna al tetraedro. Nótese, que ésta es la única nueva arista que no está en la frontera del tetraedro que se divide.

La figura 4.13 muestra un ejemplo de refinamiento en dimensión dos. Allí, los nodos N_1, N_2 y N_3 son propios en τ_j y los nodos A, B, C y D son heredados en τ_j. Por otro lado, las aristas f_1 y f_2 son externas en τ_j, mientras que f_3 es interna y c_1 es heredada.



(a) Malla τ_{j-1} (b) Malla τ_j

Figura 4.13: Ejemplo de refinamiento

Algunas propiedades de las secuencias de triangulaciones anidadas o encajadas en 2D y 3D, son las siguientes:

i) Toda arista, cara o elemento propio de cualquier triangulación es heredado en la triangulacion siguiente o tiene sus hijos en ella. Es decir, las familias de aristas, caras y tetraedros relativas a las triangulaciones no están encajadas como ocurre con los nodos.

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

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

Esta última propiedad es esencial para el algoritmo de desrefinamiento tanto en el plano como en el espacio. Es importante destacar que, como se verá más adelante, la malla más fina va cambiando en el proceso de desrefinamiento y por eso tienen sentido las propiedades anteriores. De esta forma, un elemento perteneciente a un nivel de malla intermedio, será susceptible de ser eliminado al desrefinar, si y sólo si, previamente, han sido eliminados todos sus sucesores.

El criterio de Desrefinamiento

En las condiciones de desrefinamiento programadas, se compara la solución numérica en cada grado de libertad por nodo, con la solución interpolada de la solución en los extremos de su *arista entorno* (ver definición de *arista entono* en la página 40, definición 2.1.5). Hemos utilizado la diferencia relativa y la diferencia absoluta entre estos dos valores. Si esta diferencia es menor que un cierto valor umbral o tolerancia, ϵ, que fija el usuario en la entrada de datos del problema, el nodo puede ser eliminado y en caso contrario no. Se puede observar que se comparan dos soluciones aproximadas, no la solución aproximada con la solución exacta, ya que ésta última es en general desconocida. Sin embargo, la idea de la condición de desrefinamiento es que si la solución interpolada es de por sí una buena aproximación de la última solución numérica, nos quedamos con la primera.

En la figura 4.14, a la izquierda se muestra un nodo propio K y su *arista entorno* con nodos extremos en $K-1$ y $K+1$. Para decidir si se puede eliminar el nodo K o no, se debe comprobar si para todos los grados de libertad por nodo se cumple la condición:

$$\left| u_h^l(K) - u_i^l(K) \right| \leq \epsilon \tag{4.3}$$

o la condición:

$$\left|u_h^l(K) - u_i^l(K)\right| \leq \epsilon \left|u_h^l(K)\right| \tag{4.4}$$

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

$$u_i^l(K) = \frac{u_h^l(K-1) + u_h^l(K+1)}{2} \tag{4.5}$$

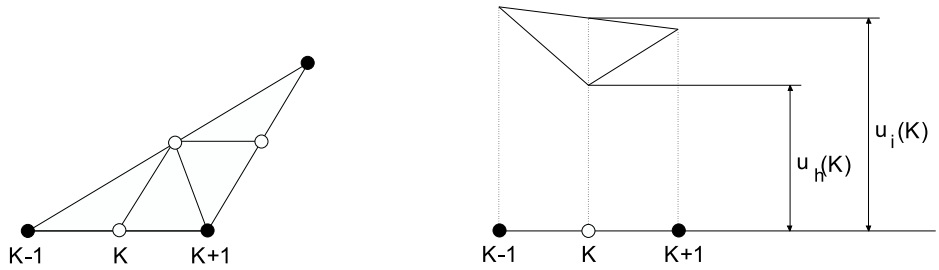


Figura 4.14: La condición de desrefinamiento

La condición expresada en la ecuación (4.4) es usada en el ámbito de los elementos finitos al tratar con soluciones numéricas.

El algoritmo de desrefinamiento 2D-SBD

Presentamos el algoritmo de desrefinamiento basado en el esqueleto 2D-SBD. En general tenemos una secuencia de mallas encajadas de la forma: $\mathbf{T} = \{\tau_1 \leq \tau_2 \leq \ldots \leq \tau_n\}$ y queremos obtener otra, tras desrefinar $\mathbf{T}$, $\mathbf{T}' = \{\tau_1 \leq \tau_2' \leq \ldots \leq \tau_m'\}$, es decir, que se cumple:

i) $m \leq n$

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

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

Un esquema del algoritmo de desrefinamiento es el que sigue:

Algoritmo 2D-SBD(T)
/* ENTRADA: Secuencia $T = \{\tau_1 < \tau_2 < \ldots < \tau_n\}$
/* SALIDA: Secuencia $T^m = \{\tau_1 < \tau_2' < \ldots < \tau_m'\}$
1: Para $j = n$ **hasta** 2 **hacer**
    Para cada nodo *propio elegible* $N \in \tau_j$, **hacer**
        /* Se calcula el conjunto $L \in$ 1-esqueleto con las
        /* aristas que han de refinarse en el paso 2
        Se examina la condición de desrefinamiento
        Se marcan los nodos a eliminar
        1.1 Se comprueba la conformidad local
        1.2 Si un nodo no se elimina, tampoco los de la arista entorno
    Fin para
Fin para
2: Para $j = 2$ **hasta** n **hacer**
    /* Se redefine la malla τ_j
    $\tau_j = 2D - SBR(\tau_{j-1}, L)$
Fin para
Fin algoritmo

El algoritmo de desrefinamiento desarrollado en esta tesis, aunque es análogo al de [66] en el sentido de que ante una entrada específica produce la misma salida al final, tiene diversas características que lo diferencia de la versión anterior. A continuación presentamos la siguiente comparativa entre los dos algoritmos:

1. La estructura de datos para almacenar la malla pasa de ser de (lista de nodos + lista de aristas en función de los nodos + lista de triángulos en función de las aristas) a esta otra, considerada como óptima, (lista de nodos + lista de triángulos en función de nodos).

2. La estructura de datos usada para implementar la partición 4T-LE en el algoritmo usa la estructura de datos del grafo G^1.

3. Se reducen las líneas de código. La nueva versión implementada en Matlab reduce considerablemente el número de líneas de código de la versión de Plaza, la cual es código Fortran.

4. La simplicidad de la nueva versión queda manifiesta, aparte de por el reducido número de líneas de código, por la flexibilidad de introducirlo

en diversas aplicaciones. Esta característica es en parte debido a la modularidad que ofrece el lenguaje Matlab. Por ejemplo, se incorpora en el lenguaje de realidad virtual $VRML$ para generar niveles de detalle, ver aplicaciones.

5. El nuevo algoritmo cambia la forma de proceder respecto al anterior. Se puede decir que la versión aquí presentada actúa de forma **forward**, ya que parte del nivel de malla más grosera hasta obtener la malla más fina. Esto puede verse como un proceso de reconstrucción de la malla en el que el refinamiento es selectivo y de acuerdo con el concepto de desrefinamiento de la malla. Por contra, el algoritmo de [66] es tipo **backward**, ya que procede a la inversa, desde el nivel de mallas más fino a la más grosera.

6. La portabilidad del nuevo algoritmo se garantiza debido a las posibilidades de traducción de código Matlab a por ejemplo, lenguaje C o Java.

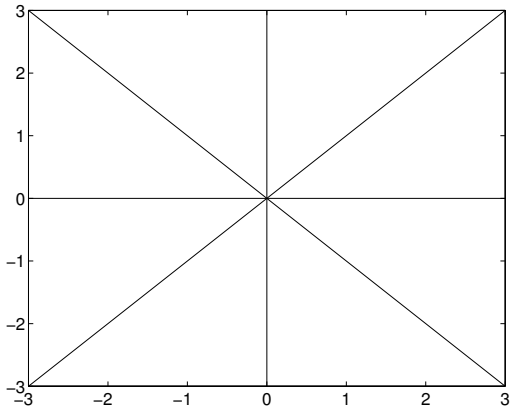
7. La complejidad lineal del algoritmo en términos de tiempo permanece, sin embargo, la complejidad en cuanto al almacenamiento se reduce y se mejora, ya que no se almacena la lista de aristas de la malla.

Para ilustrar el uso del algoritmo de desrefinamiento 2D-SBD, presentamos un ejemplo que consiste en el desrefinamiento de 4 niveles de mallas encajadas. El criterio de desrefinamiento es por diferencia absoluta en los nodos con tolerancia de 0,1875. El ejemplo de desrefinamiento aproxima la singularidad de una función matemática en el punto $(-1, 0)$. En las las figuras 4.15 y 4.16 se ilustran los cuatro niveles de mallas antes de desrefinar y después de desrefinar (el símbolo asterisco indica que el nodo es eliminado en ese nivel de desrefinamiento).

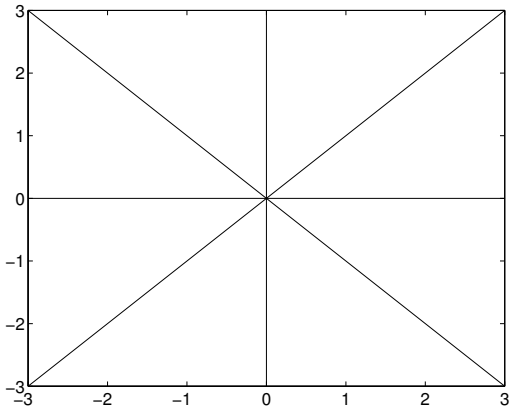
Para completar en dimensión 3 con un algoritmo de desrefinamiento y aplicaciones, emplazamos a las referencias [63, 72, 73].

4.6. Propiedades geométricas de la partición 4T-LE

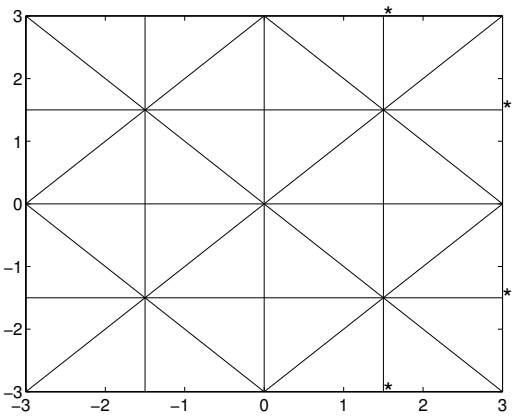
En esta sección se estudian algunas propiedades geométricas interesantes de la partición 4T-LE para mallas bidimensionales triangulares. Los algoritmos



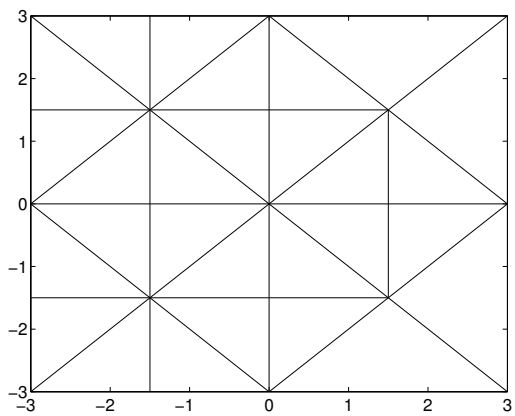
(a) Malla 1, p=9, t=8



(b) Malla desrefinada, p=9, t=8

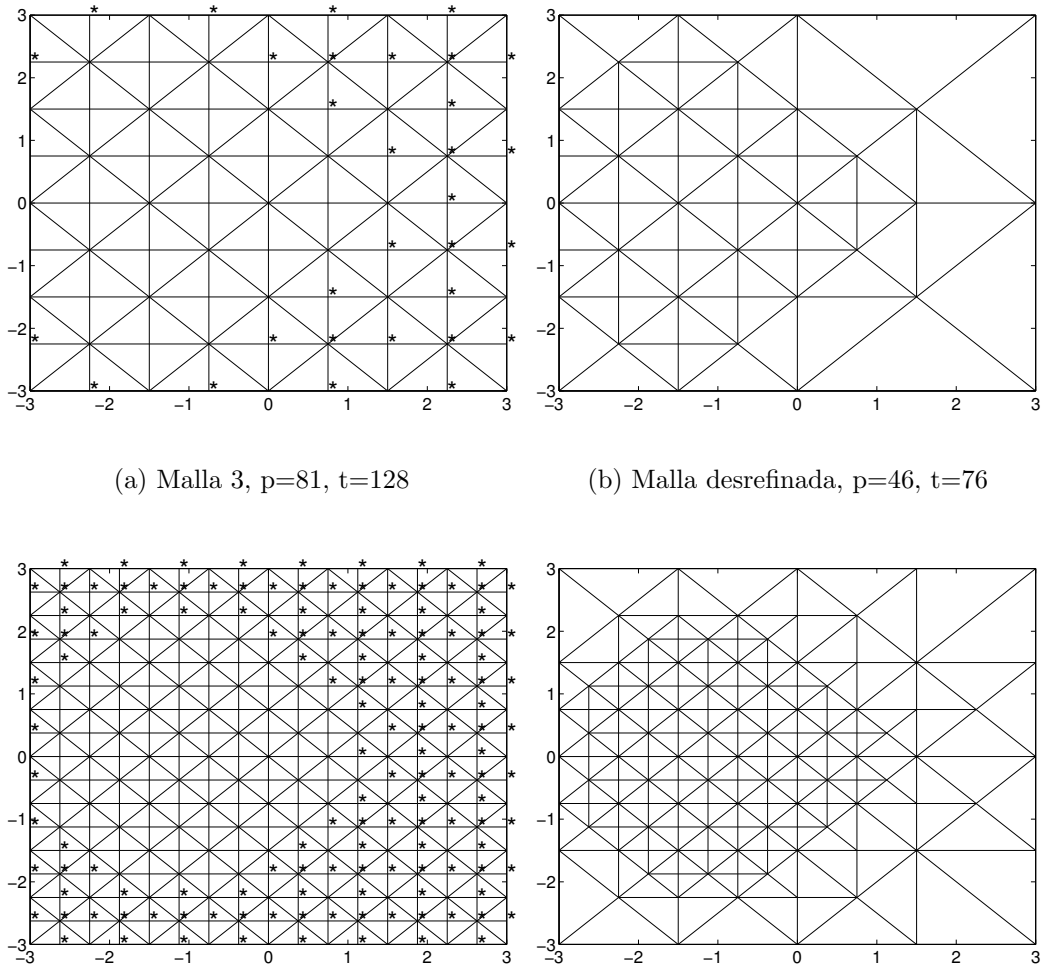


(c) Malla 2, p=25, t=32



(d) Malla desrefinada, p=21, t=28

Figura 4.15: Mallas 1 y 2 del test del algoritmo 2D-SBD



(a) Malla 3, p=81, t=128

(b) Malla desrefinada, p=46, t=76

(c) Malla 4, p=289, t=512

(d) Malla desrefinada, p=131, t=242

Figura 4.16: Mallas 3 y 4 del test del algoritmo 2D-SBD

de refinamiento y desrefinamiento presentados anteriormente son útiles para descubrir empíricamente propiedades del refinamiento de mallas triangulares. Asimismo, la introducción del grafo del esqueleto, que está íntimamente relacionado con el *LEPP* de Rivara motiva este tipo de estudio, ya que permite modelar fácilmente los procesos de refinamiento en los que la conformidad y la partición se definen por el lado mayor de los triángulos. Se dan primeramente un conjunto de definiciones y preliminares.

4.6.1. Definiciones y preliminares

Definición 4.6.1 (Triángulo interior y triángulo exterior) *Sea τ una malla triangular n-dimensional (n = 2 o 3) de un dominio acotado Ω. Un triángulo t $\in \tau$ se dice que es exterior o de frontera si tiene una (n-1)-cara en la frontera de Ω. En caso contrario se dice que es un triángulo interior.*

Nótese que el *LEPP* de un triángulo dado t (ver página 104, definición 4.1.1) empieza por el propio triángulo t. Obsérvese también que el *LEPP* de un triángulo dado t termina o bien en un triángulo con el lador mayor en la frontera del dominio, o bien en un par de triángulos que comparten la arista mayor. En ambos casos se llaman *triángulos terminales*. En el primer caso se dice que es un triángulo terminal de frontera o externo y en el segundo que son un par de triángulos terminales interiores.

Definición 4.6.2 (Malla equilibrada y malla equilibrada estable) *Sea τ una triangulación bi-dimensional. Se dice que τ es equilibrada si está compuesta de triángulos terminales. Además se llama malla equilibrada estable si la malla es equilibrada y la aplicación de la partición 4T-LE a todos los triángulos de la malla produce otra malla equilibrada.*

Definición 4.6.3 (Malla semi-equilibrada) *Se dice que τ es una malla semi-quilibrada si está compuesta de triángulos terminales y tiene además algunos triángulos que no forman parte de pares de triángulos terminales.*
El grado de equilibrio de una malla semi-equilibrada, denotada por $gr(\tau)$, viene dado por el cociente entre el número de triángulos terminales (TT) y el número total de triángulos (T).

$$gr(\tau) = \frac{|TT|}{T} \tag{4.6}$$

Si el grado de equilibrio de una malla es 1, entonces la malla es equilibrada. Si es aproximadamente 1, la malla es semi-equilibrada. En una malla que no es equilibrada y que no tiene ningún par de triángulos terminales, el refinamiento de un elemento cualquiera se extiende por conformidad hasta alcanzar la frontera del dominio.

En la figura 4.17 se muestran una malla equilibrada (a), y una semi-equilibrada (b). Además aparecen sombreados los triángulos irregulares (que no son triángulos terminales).

Un ejemplo de malla equilibrada estable es aquella formada por triángulos rectángulos opuestos dos a dos por la hipotenusa. La aplicación de la partición 4T-LE produce triángulos semejantes a los originales y todos ellos triángulos terminales.

4.6.2. Propiedades geométricas de la partición 4T-LE

Presentamos primeramente algunas propiedades basadas en trabajos de Rosenberg y Stenger [89] y Stynes [96] sobre el método de bisección de un triángulo también recogidas por Rivara en [79]:

Proposición 4.6.1 *Sea α_0 el menor ángulo de un triángulo* t. *Si dividimos reiteradamente* t *y los triángulos siguientes mediante la bisección por el lado mayor, y llamamos α_j al menor ángulo interior de la j-ésima triangulación así obtenida -obsérvese que no tiene por qué ser conforme-, entonces se tiene:*

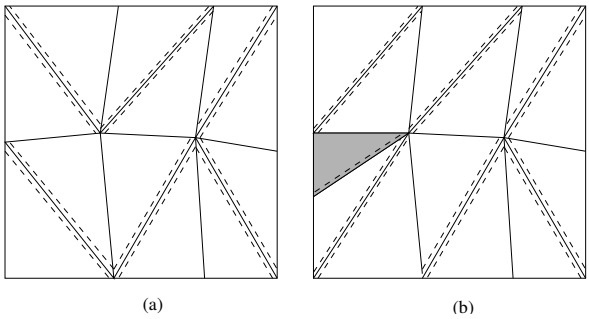


Figura 4.17: (a) Malla equilibrada, (b) malla semi-equilibrada

$$\alpha_j \geq \frac{\alpha_0}{2} \tag{4.7}$$

□

Proposición 4.6.2 *Si h_0 es el diámetro -esto es, longitud del lado mayor- del triángulo inicial* t*, y h_j el diámetro de la malla obtenida tras j refinamientos, entonces existe un entero positivo N, tal que:*

$$h_{2j} \leq \sqrt{3}^{min(j,N)} \left(\frac{1}{2}\right)^j h_0 \tag{4.8}$$

□

De la proposición 4.6.1, Rivara deduce que para triangulaciones obtenidas de la aplicación de sus algoritmos 2-T o 4-T a una triangulación inicial, se tiene la misma relación entre sus ángulos mínimos, es decir

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

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

$$\frac{\text{mín}(h_1, h_2)}{\text{máx}(h_1, h_2)} \geq \delta > 0 \tag{4.9}$$

La ecuación (4.7) hace referencia a la no degeneración de la malla, mientras que la ecuación (4.9) nos da información sobre la gradualidad de la discretización. Estas dos propiedades le permiten a Rivara hablar de triangulaciones que pertenecen a la misma familia de triangulaciones conformes, ya que las constantes α_0 y δ dependen de la triangulación inicial y caracterizan tales familias de triangulaciones.

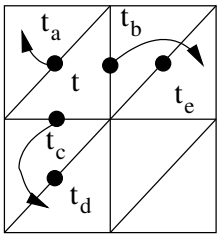
Recogemos a continuación algunas propiedades geométricas de la partición 4T-LE relativas a la forma de los triángulos generados por la partición. Veremos que estas propiedades están relacionadas con la localidad del algoritmo de refinamiento asociado.

La siguiente propiedad tomada de la referencia [87] está en relación con los triángulos semejantes distintos que se generan al aplicar la partición 4T-LE.

Proposición 4.6.3 *(a) La primera partición 4T-LE de cualquier triángulo t_1 produce dos nuevos triángulos semejantes a t_1 (exteriores a t_1 por su arista mayor y dos triángulos semejantes entre sí y posiblemente no semejantes a t_1. (b) La aplicación reiterada de la partición 4T-LE a cualquier triángulo inicial t_1 produce como máximo un triángulo distinto (en cuanto a semejanza) por cada iteración.* □

Corolario 4.6.1 *Si en una malla equilibrada, τ , la partición 4T-LE produce en cada triángulo un par de triángulos terminales interiores, entonces τ es una malla equilibrada estable.* □

Para el análisis de las propiedades geométricas de la partición 4T-LE y los algoritmos de refinamiento asociados, utilizaremos dos métricas. La primera, que llamaremos $M1$, es el número adicional de triángulos refinados al refinar t. La segunda métrica, que llamaremos $M2$, es la longitud máxima de los *LEPP's* de los vecinos de t. Obsérvese que la primera métrica coincide con la suma de las longitudes de los *LEPP* de los triángulos vecinos de t, sin contar el propio triángulo t en esos caminos de propagación. Véase el ejemplo de la figura 4.18.



Triángulo	LEPP
t	{t,t$_a$}
t$_a$	{t$_a$,t}
t$_b$	{t$_b$,t$_e$}
t$_c$	{t$_c$,t$_d$}

Figura 4.18: Extensión por conformidad y LEEP de triángulos vecinos

Lema 4.6.1 *Sea τ una malla equilibrada. Entonces para todo triángulo interior de τ $M1 = 5$ y $M2 = 2$, ver figura 4.18. Además obsérvese que $M1 = 5$ si y sólo si $M2 = 2$, y que cualquiera de esas condiciones equivalen a que la malla sea equilibrada.* □

Nuestro objetivo es demostrar que la aplicación global (y también local en cierto sentido que se explicará) hace que las mallas que aparezcan sean cada vez más equilibradas. Para ello, por la observación anterior, demostraremos

que asintóticamente, la media de la métrica $M1$ tiende a 5 cuando el número de aplicaciones de la partición 4T-LE tiende a infinito. Para ello además, basta considerar el caso de un único triángulo como malla inicial, pues el que una malla sea o no equilibrada depende de si los pares de triángulos terminales cubren todo el dominio inicial o no. Además como estamos interesados en el comportamiento asintótico de la partición 4T-LE, cuando hallemos triángulos semejantes a otros triángulos estudiados previamente, les podremos aplicar los mismos argumentos. Como se ve, la propiedad que demostraremos está relacionada con el número de clases de triángulos distintos (en cuanto a semejanza) que pueden aparecer en la aplicación de la partición 4T-LE a una triangulación (un triángulo) inicial. Comencemos por un caso sencillo.

Proposición 4.6.4 *(Caso de triángulo rectángulo). La aplicación de la partición 4T-LE a un triángulo rectángulo t_1 produce 4 triángulos semejantes a t_1, de los cuales los dos triángulos no exteriores por la hipotenusa son terminales.* □

Corolario 4.6.2 *La aplicación reiterada de la partición 4T-LE a un triángulo rectángulo t produce mallas de triángulos semejantes con el original en las que el área cubierta por pares de triángulos terminales tiende a ser el área del triángulo original.* □

Proposición 4.6.5 *(Caso de triángulo acutángulo). La aplicación de la partición 4T-LE a un triángulo t con los tres ángulos agudos produce dos triángulos semejantes al original sobre el lado mayor del original, y un par de triángulos terminales semejantes entre sí, no exteriores por el lado mayor del triángulo original.* □

Corolario 4.6.3 *La aplicación reiterada de la partición 4T-LE a un triángulo acutángulo t produce mallas de triángulos en las que el área cubierta por pares de triángulos terminales tiende a ser el área del triángulo original.* □

El corolario 4.6.3 se deduce directamente de la proposición 4.6.5 y de que sólo se producen dos triángulos similares distintos (ver por ejemplo [87], página 1489).

El caso más complicado es el del triángulo obtusángulo. En este caso se pueden distinguir dos situaciones que se analizan en la siguiente proposición:

Proposición 4.6.6 *(Caso de triángulo obtusángulo). La aplicación de la partición 4T-LE a un triángulo obtusángulo t produce 2 triángulos semejantes al original sobre el lado mayor del original y, o bien:*

1. *un par de triángulos terminales semejantes entre sí no exteriores, (se tiene una situación igual a la de la figura 4.19), o bien,*

2. *un par de triángulos semejantes entre sí no terminales, (ver figura 4.20).*

□

Corolario 4.6.4 *Si la partición 4T-LE produce en un triángulo obtusángulo la situación del apartado 1 de la proposición 4.6.6, entonces la aplicación reiterada de la partición produce mallas de triángulos en las que el área cubierta por pares de triángulos terminales tiende a ser el área del triángulo original.*

Prueba
La consecuencia anterior se deduce directamente de la proposición 4.6.6 y de que sólo se producen dos triángulos similares distintos (ver por ejemplo [87], página 1489). □

Para demostrar la consecuencia análoga a la anterior para el caso 2 de la proposición 4.6.6 hay que considerar la situación que refleja el siguiente teorema debido a Rivara e Iribarren [87]:

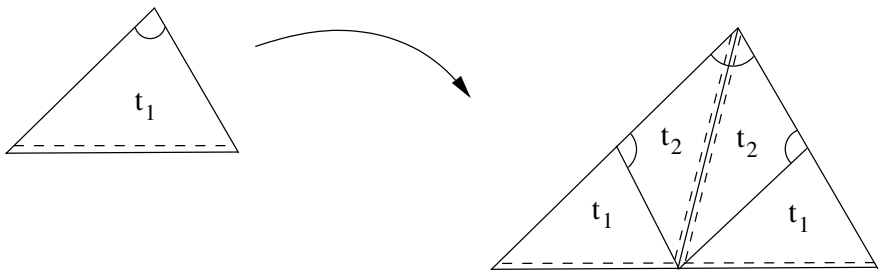


Figura 4.19: 4T-LE en un triángulo acutángulo

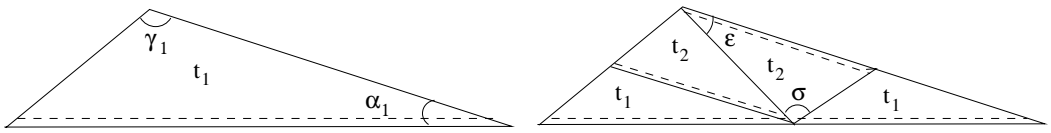


Figura 4.20: (a) 4T-LE en un triángulo obtusángulo (b) Partición 4T-LE

Teorema 4.6.1 *Si la partición 4T-LE produce en un triángulo obtusángulo la situación del apartado 2 de la proposición 4.6.6, entonces los ángulos del nuevo triágulo t_2 generado verifican las siguientes relaciones, ver Figura 4.20:*

1. $\gamma_2 = \sigma$

2. $\gamma_2 = \gamma_1 - \varepsilon \leq \gamma_1 - \alpha_1$

donde γ_i es el ángulo mayor del triángulo t_i y α_1 es el ángulo menor de t_1. □

Nótese que el apartado 2 del teorema significa que el nuevo triángulo es un triángulo *mejor* que el triángulo original (menos obtuso). Reiterando el proceso (la partición 4T-LE) se llega a un triángulo que o bien, cumple el apartado 1 de la proposición anterior, o bien es no obtuso, con lo que se le pueden aplicar las proposiciones previas.

Corolario 4.6.5 *El número de triángulos semejantes distintos generados al aplicar la partición 4T-LE sobre un triángulo inicial cualquiera está acotado, y depende sólo de la geometría del triángulo inicial. En concreto, el número de triángulos distintos generados, N, cumple la siguiente expresión:*

$$N \leq \left\lceil \left| \frac{\gamma_1 - \frac{\pi}{2}}{\alpha_1} \right| \right\rceil + 1 \tag{4.10}$$

donde γ_1 es el ángulo mayor del triángulo inicial y α_1 es el ángulo menor.

Prueba

Aplicando por recurrencia el apartado 2 del teorema 4.6.1 se tiene que $\gamma_N \leq \gamma_1 - N\alpha_1$ lo que es válido siempre que tengamos un triángulo obtuso. De ahí se tiene el resultado. □

Nótese que la expresión anterior sobre una cota del número de triángulos semejantes generados mejora la anterior cota de Rivara e Iribarren:

$$N \leq \left\lceil \frac{\gamma_1 - \frac{\pi}{3}}{\alpha_1} \right\rceil + 1. \tag{4.11}$$

Además la nueva cota propuesta es ajustada para el caso de triángulos rectángulos y válida también para triángulos acutángulos.

En la tabla 4.2 se presenta una comparativa entre la cota de Rivara-Iribarren [87] sobre el número de triángulos distintos (en cuanto a semejanza) generados y la nueva cota aquí presentada, corolario 4.6.5. El experimento consiste en la aplicación sucesiva de la partición 4T-LE a un triángulo obtusángulo inicial. Se selecciona cada vez para seguir dividiéndolo, el triángulo que no sea semejante al triángulo previo a la división. Con ello se aplica sucesivamente la partición en la clase de triángulos de tipo obtusángulos y se observa cómo en cada iteración, el ángulo mayor representado por γ tiende a 90 grados. Se interrumpen las iteraciones cuando se generan triángulos que empiezan a ser acutángulos. Se representan además la medida de los ángulos interiores, siendo γ el ángulo mayor de cada triángulo, β y el ángulo mediano de cada triángulo y α el ángulo menor. En cada uno de los seis casos presentados, se dan las medidas de ambas cotas. Nótese cómo en efecto se produce la mejora, lo que está en concordancia con la ecuación (4.11). El experimento refleja también la propiedad de automejora de la partición 4T-LE. Recientes estudios ofrecen mejoras sobre las cotas de los triángulos generados y sobre la mejora que produce la partición 4T-LE [67].

Cuadro 4.2: Secuencia de triángulos distintos obtenidos por la aplicación iterativa de la partición 4T-LE

	Triángulo 1			Triángulo 2		
	# triángulos distintos		15	# triángulos distintos		11
	Cota R-I		45	Cota R-I		190
	Nueva Cota		30	Nueva Cota		140
It.	γ	β	α	γ	β	α
0	145.455	32.595	1.950	173.972	5.423	0.605
1	143.292	34.545	2.164	173.216	6.028	0.756
2	140.885	36.708	2.407	172.245	6.784	0.971
3	138.200	39.115	2.684	170.952	7.755	1.293
4	135.202	41.800	2.998	169.148	9.048	1.804
5	131.850	44.798	3.351	166.462	10.852	2.686
6	128.107	48.150	3.743	162.066	13.538	4.396
7	123.937	51.893	4.170	153.735	17.934	8.331
8	119.316	56.063	4.621	133.923	26.265	19.812
9	114.235	60.684	5.081	84.274	49.648	46.077
10	108.715	65.765	5.520	95.726	43.599	40.676
11	102.811	71.285	5.904	84.274	49.648	46.077
12	96.618	77.189	6.193			
13	90.266	83.382	6.352			
14	89.734	83.907	6.359			
15	90.266	83.382	6.352			

	Triángulo 3			Triángulo 4		
	# triángulos distintos		8	# triángulos distintos		4
	Cota R-I		73	Cota R-I		7
	Nueva Cota		54	Nueva Cota		4
It.	γ	β	α	γ	β	α
0	169.900	8.572	1.527	114.624	54.900	10.475
1	167.721	10.100	2.180	102.073	65.376	12.551
2	164.371	12.279	3.349	88.250	77.927	13.824
3	158.625	15.629	5.747	91.750	74.623	13.627
4	146.921	21.375	11.704	88.250	77.927	13.824
5	117.268	33.079	29.652			
6	63.237	62.732	54.031			
7	116.763	33.270	29.967			
8	63.237	62.732	54.031			

	Triángulo 5			Triángulo 6		
	# triángulos distintos		3	# triángulos distintos		2
	Cota R-I		5	Cota R-I		5
	Nueva Cota		3	Nueva Cota		2
It.	γ	β	α	γ	β	α
0	130.542	27.127	22.332	116.565	45.000	18.435
1	76.437	54.105	49.458	90.000	63.435	26.565
2	103.563	39.659	36.777	90.000	63.435	26.565
3	76.437	54.105	49.458			

Corolario 4.6.6 *Si la partición 4T-LE produce en un triángulo obtusángulo la situación del apartado 2 de la proposición 4.6.6, entonces la aplicación reiterada de la partición produce mallas de triángulos en las que el área cubierta por triángulos terminales tiende a ser el área del triángulo original.*

Prueba
Para demostrar esta consecuencia basta considerar la mejora que produce la partición 4T-LE sobre la triangulación y tener en cuenta que cuando se tiene la situación 1 de la proposición 4.6.6 entonces sólo se generan triángulos terminales. □

Corolario 4.6.7 *La partición 4T-LE produce al ser aplicada un número suficiente de veces sobre una triangulación incial τ_0, una malla que es semiequilibrada y estable τ_n. Más aún:*

$$\lim_{n\to\infty} gr(\tau_n) = 1 \tag{4.12}$$

□

Nótese que se tiene entonces que la medias de las medidas $M1$ y $M2$ introducida previamente son tan cercanas a 5 y a 2 respectivamente como se quiera. Los ejemplos numéricos de la siguiente sección están en pleno acuerdo con lo anterior.

Corolario 4.6.8 *La media de los ángulos mayores tiende a 90^o al aplicar la partición 4T-LE un número suficiente de veces sobre una triangulación incial.*

Prueba
El corolario 4.6.7 asegura que la aplicación reiterada de la partición 4T-LE tiende a hacer la malla más equilibrada y estable, es decir que el área cubierta por triángulos terminales tiende a ser el área del dominio inicial. Por tanto la propiedad sobre la media del ángulo máximo de los triángulos queda demostrada si lo hacemos para una malla simple compuesta por un par de triángulos terminales estable, es decir en la que la aplicación de la partición 4T-LE provoca la aparición de triángulos terminales. (Véase figura 4.21).

Nótese que si llamamos γ_i al ángulo máximo del triángulo t_i, para i=1,2, en la figura 4.21, al aplicar la partición 4T-LE la media de los ángulos máximos

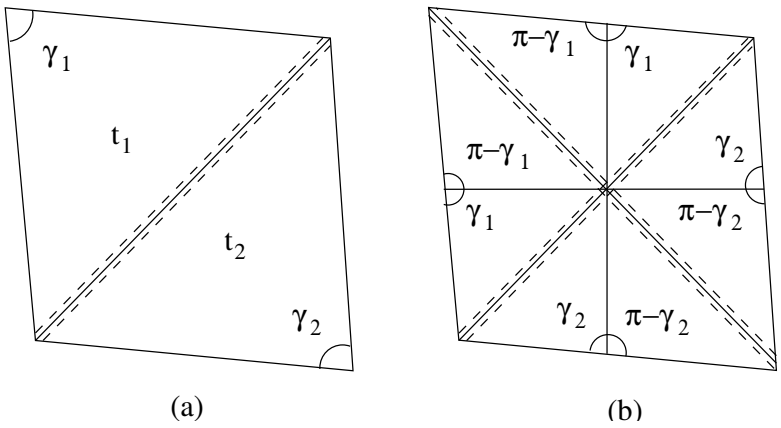


Figura 4.21: (a) Triángulos terminales estables (b) Partición 4T-LE

es 90 grados como se quería demostrar.

En el ejemplo de la figura 4.21 (a) se aplica la partición 4T-LE y resulta la nueva malla de la figura 4.21 (b). Se observa que los triángulos después de la aplicación de la partición vuelven a ser triángulos terminales y la media de los angulos máximos es $\frac{\pi}{2}$. □

Podría parecer que se puede deducir una propiedad similar para el comportamiento asintótico de la media del ángulo mínimo, pero esto no es posible como se puede observar en el siguiente ejemplo.

Ejemplo: Sea τ una triangulación compuesta por un único triángulo rectángulo. Como la aplicación de la partición 4T-LE a un triángulo rectángulo sólo genera triángulos rectángulos semejantes al original, es claro que la media de los ángulos mínimos generados es constante e igual al ángulo mínimo del triángulo inicial. Es claro que se puede elegir la malla inicial con ángulo mínimo medio arbitrario y por tanto no tenemos una propiedad semejante a la anterior respecto del ángulo mínimo. Al menos no tenemos una propiedad relativa al ángulo mínimo que sea independiente de la malla elegida.

4.6.3. Experimentos con el algoritmo 2D-SBR

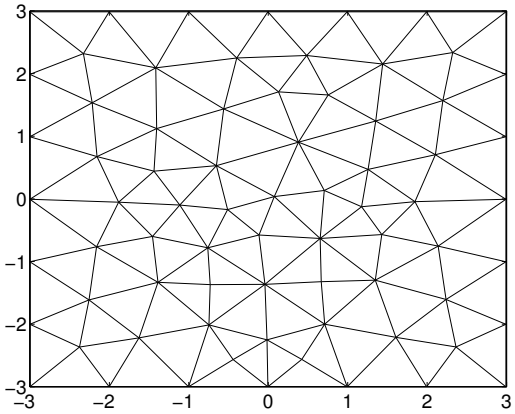
A continuación se presentan dos problemas test. Se trata de aplicar el algoritmo 2D-SBR recién presentado con la partición 4T-LE a dos triangulaciones iniciales.

La primera triangulación, ver figura 4.22 (a), considerada como buena, es de tipo Delaunay y conserva las buenas propiedades de dicho tipo de triangulaciones. Para dicha malla, buena parte de los triángulos son triángulos terminales, ver tabla 4.5 en la página 153.

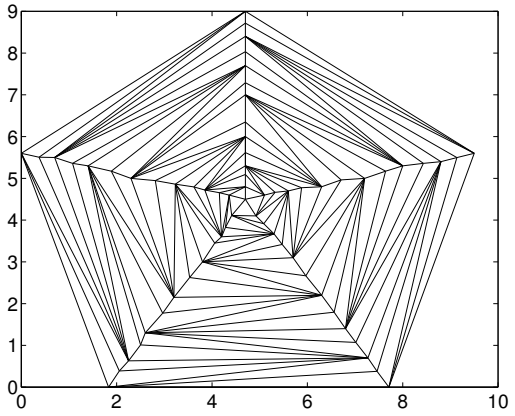
El segundo experimento considera una malla en la que no hay ningún par de triángulos terminales, y además todos los triángulos son malos desde el punto de vista de sus ángulos, es decir, con ángulos máximos grandes y ángulos mínimos pequeños. Nótese que se puede construir una malla tan mala como se quiera (respecto de las medidas $M1$ y $M2$ introducidas anteriormente) siguiendo la idea del ejemplo. Llamamos a esta malla, malla pentagonal, ver figura 4.22 (b).

La tabla 4.3 y la figura 4.23 en relación con la malla Delaunay, muestra cómo evolucionan las medidas $M1$ y $M2$ al aplicar sucesivamente la partición 4T-LE a todos los triángulos del nivel de malla anterior, cuatro niveles de refinamiento. Nótese que rápidamente $M1$ tiende a 5 y $M2$ a 2, como se ha demostrado anteriormente. Como era de esperar debido a las buenas características de este tipo de mallados, la métrica $M1$ pasa de ser 5.2333 a 5.0145, con lo cual la tendencia a 5 de $M1$ en este caso se consigue de forma rápida. Lo mismo ocurre para $M2$ en la misma malla Delaunay, evolucionando de 2.6250 a 2.0481.

De forma análoga se presenta en la tabla 4.4 y la figura 4.24 los datos de $M1$ y $M2$ para la malla pentagonal con cuatro niveles de refinamiento. Puede notarse cómo la métricas $M1$ al inicio de la malla pentagonal se eleva a más de 26 en media. Tras aplicar el primer nivel de refinamiento, dicha media baja a cerca de 7, lo que revela la reducción notable que sufre la métrica $M1$ y ello redunda en la calidad de los elementos generados. Después del nivel refinamiento número cuatro la métrica $M1$ pasa a ser 5.4823. De la misma forma puede verse la reducción de la métrica $M2$ en este ejemplo, pasando de 14.3929 en el



(a) Malla tipo Delaunay



(b) Malla pentagonal

Figura 4.22: Mallas de los problemas test

nivel inicial a 2.4123 en el nivel número cuatro. Nótese finalmente la diferencia en comparativa de las dos mallas teniendo en cuenta que el número de refinamientos aplicados son los mismos, y que el número de triángulos generados en cada nivel son del mismo orden.

El siguiente dato que se proporciona para estos dos problemas test es el estudio del grado de equilibrio, ver corolario 4.6.3, página 136. En las tablas 4.5 y 4.6 se presentan la evolución numérica del grado de equilibrio para la malla Delaunay y pentagonal respectivamente. Se observa que la malla Delaunay, partiendo de un grado de equilibrio de 0.48333, llega en el nivel de refinamiento cuatro a ser aproximadamente 1, lo que indica que las mallas generadas tienden a ser más equilibradas en cada paso de refinamiento. Dicho efecto también se puede apreciar en la malla pentagonal, donde partiendo de una grado de equilibrio 0, pasa a ser en el refinamiento cuatro, de 0.66375, lo que también muestra que aumenta el equilibrado de la malla. En la figura 4.25 se ofrece una gráfica comparativa del grado de equilibrio para los dos ejemplos.

En las tablas 4.7 y 4.8 se presentan datos de los ángulos interiores de los triángulos en cada nivel de refinamiento. Los datos de la tabla son en este orden: el número de triángulos de cada nivel de refinamiento, la media de los valores máximos de los ángulos interiores para cada triángulo generado, la media de los valores mínimos de los angulos interiores, el máximo valor de los ángulos, el mínimo valor de los ángulos, el número de ángulos que tienen un valor mayor de 120 grados y finalmente, el número de ángulos que tienen un valor menor de 20 grados. Se puede observar cómo la tendencia del ángulo máximo es a 90 grados y ello se manifiesta cláramente para la malla de Delaunay. Asimismo, el ángulo mínimo se mejora en el caso de la malla pentagonal y se estabiliza para la malla Delaunay. Se deja como línea futura, la publicación de nuevas propiedades en relación con el ángulo mínimo al utilizar la partición 4T-LE.

Las figuras 4.28, 4.29 muestran el resultado del refinamiento global sucesivo de la malla pentagonal, así como la malla resaltada con los triángulos terminales. Se observa como la aplicación del refinamiento global del algoritmo 2D-SBR produce cada vez más triángulos terminales en cada nivel de refinamiento. En particular para este ejemplo, los triángulos terminales sur-

gen desde el centro del pentágono y se extienden cada vez más hacia afuera (contorno). Para el último nivel de refinamiento (cuatro) se ha optado por presentar dos detalles de la malla original debido a la gran densidad de elementos en este nivel. El primer detalle, figura 4.29 (a) corresponde a la parte central del pentágono, donde puede observarse que practicamente todo el área representada es cubierta por triángulos terminales. El segundo detalle, figura 4.29 (b) corresponde a la parte de la derecha del pentágono, cerca del contorno. En este último caso se observa cómo la presencia de triángulos que no son terminales es notable, y sólo la iteración sucesiva del refinamiento global hará que desaparezcan, para aparecer triángulos terminales, lo que está en conjunción con lo dicho en el corolario 4.6.7.

Cuadro 4.3: Malla Delaunay. Métricas M1 y M2

Nivel Refin.	N. Triángulos	Media M1	Media M2
0 (Inicio)	120	5.2333	2.6250
1	480	5.1125	2.3813
2	1920	5.0573	2.1953
3	7680	5.0289	2.0967
4	30720	5.0145	2.0481

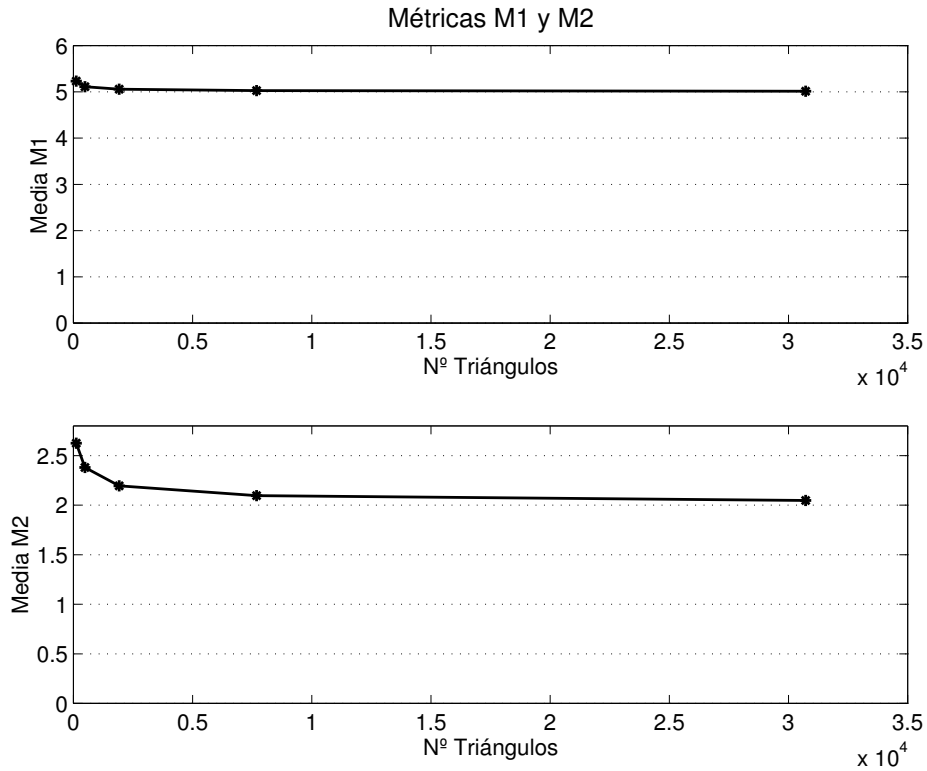


Figura 4.23: Refinamiento global 4T-LE y evolución de M1 y M2. Malla Delaunay

Cuadro 4.4: Malla pentagonal. Métricas M1 y M2

Nivel Refin.	N. Triángulos	Media M1	Media M2
0 (Inicio)	125	26.5440	14.3929
1	500	6.9100	3.8000
2	2000	6.2000	3.0485
3	8000	5.9976	2.8312
4	32000	5.4823	2.4123

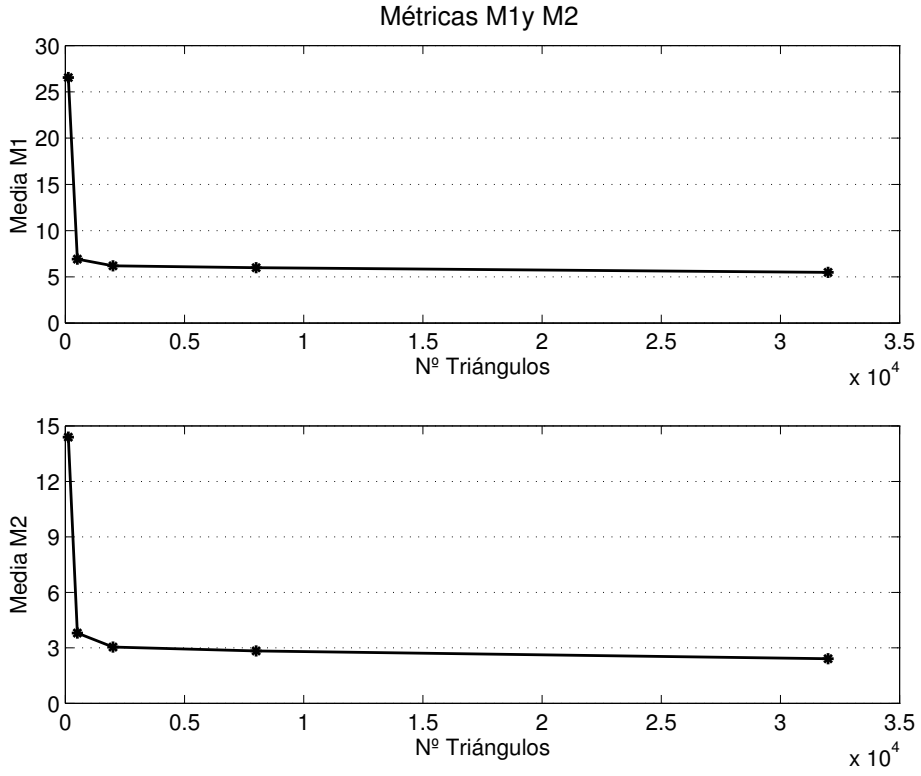


Figura 4.24: Refinamiento global 4T-LE y evolución de M1 y M2. Malla pentagonal

Cuadro 4.5: Malla Delaunay. Grado de equilibrio

Nivel Ref.	Triángulos Terminales	Triángulos totales	Grado Equilibrio
0 (Inicio)	58	120	0.48333
1	356	480	0.74166
2	1672	1920	0.87083
3	7184	7680	0.93541
4	29728	30720	0.96770

Cuadro 4.6: Malla Pentagonal. Grado de equilibio

Nivel Ref.	Triángulos Terminales	Triángulos totales	Grado Equilibrio
0 (Inicio)	0	125	0
1	246	500	0.492
2	1088	2000	0.544
3	4778	8000	0.59725
4	21240	32000	0.66375

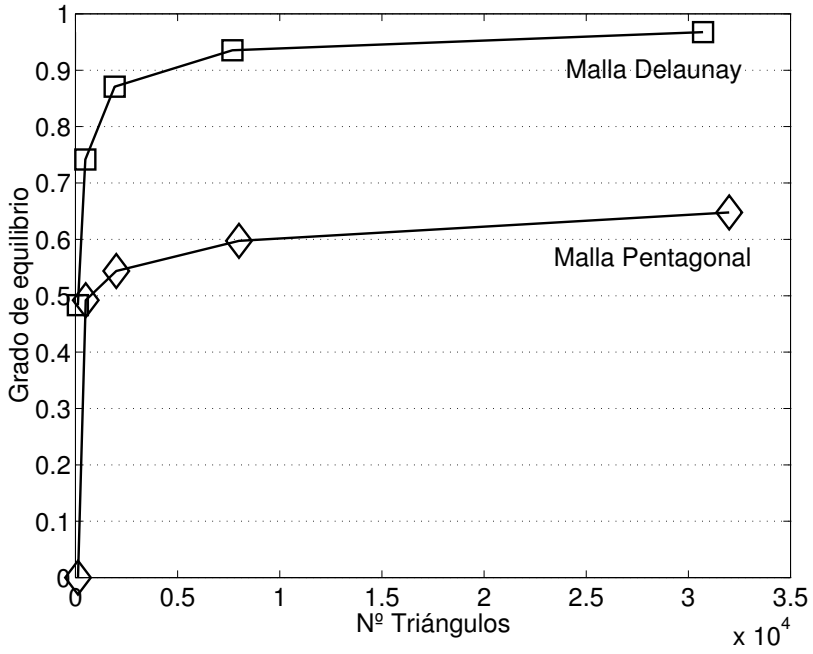
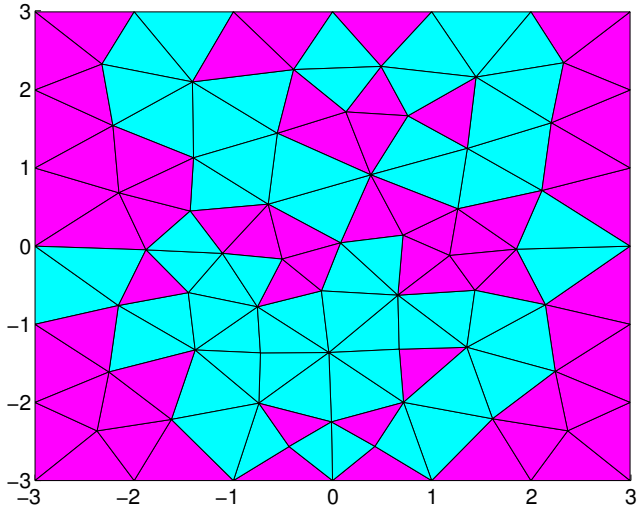
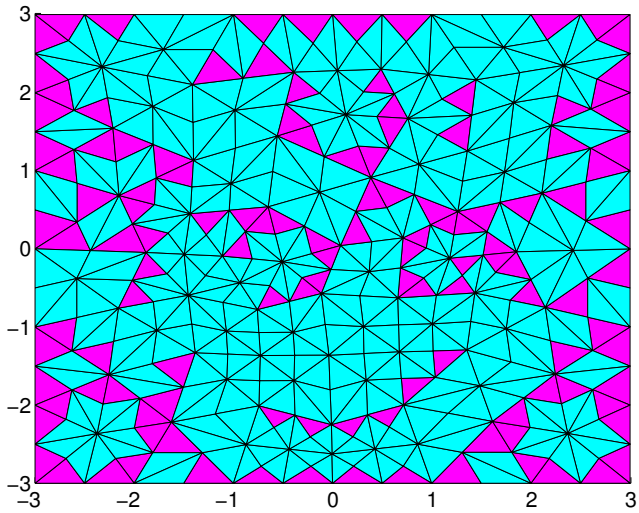


Figura 4.25: Grado de equilibrio. Malla pentagonal y Malla Delaunay

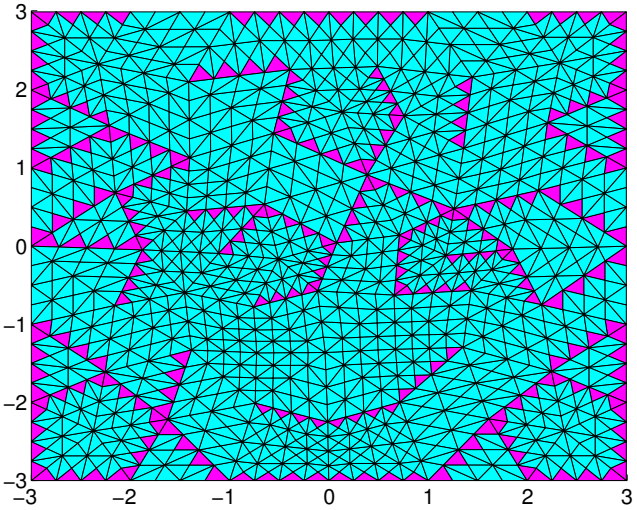


(a) Malla inicial, 120 triángulos totales, 58 triángulos terminales (color azul)

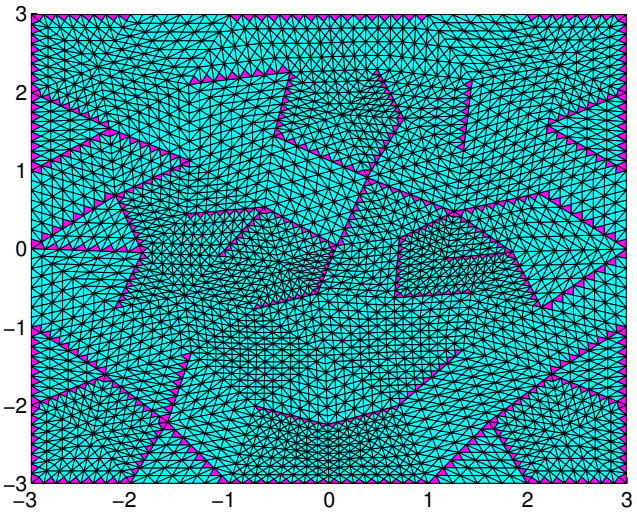


(b) Malla 1, 480 triángulos totales, 356 triángulos terminales (color azul)

Figura 4.26: Malla inicial tipo Delaunay y nivel de refinamiento uno. Refinamiento global 4T-LE

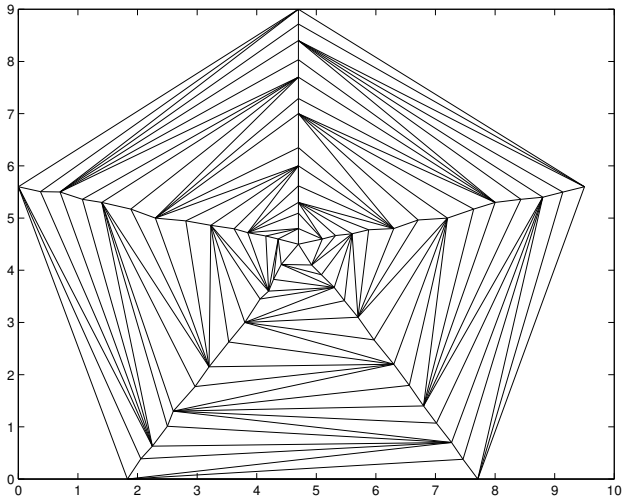


(a) Malla 2, 1920 triángulos totales, 1672 triángulos terminales (color azul)

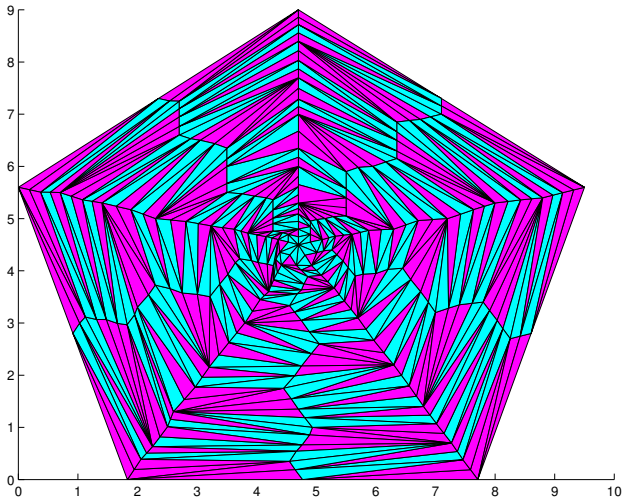


(b) Malla 3, 7680 triángulos totales, 7184 triángulos terminales (color azul)

Figura 4.27: Malla Delaunay. Niveles de refinamiento dos y tres. Refinamiento global 4T-LE

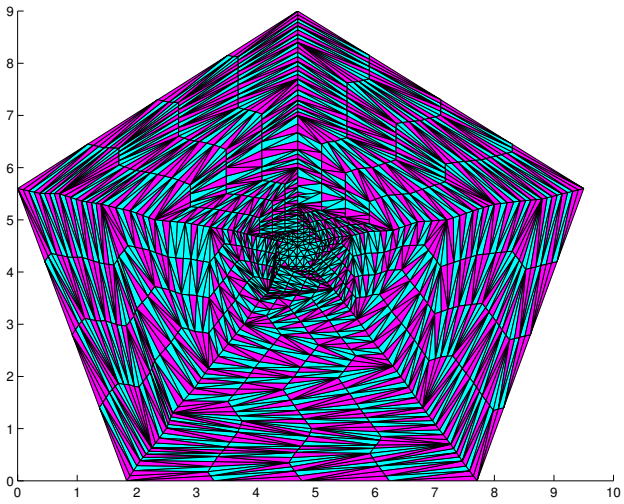


(a) Malla inicial, 0 triángulos terminales, 125 triángulos totales

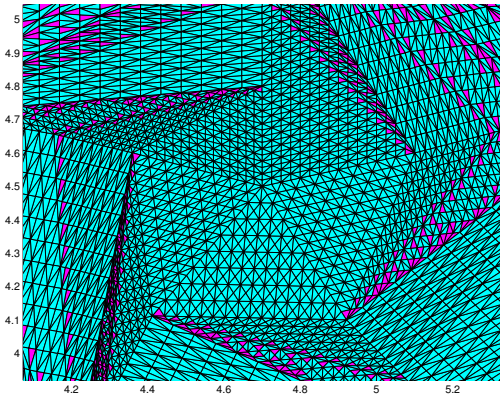


(b) Nivel de refinamiento uno, 246 triángulos terminales (color azul), 500 triángulos totales

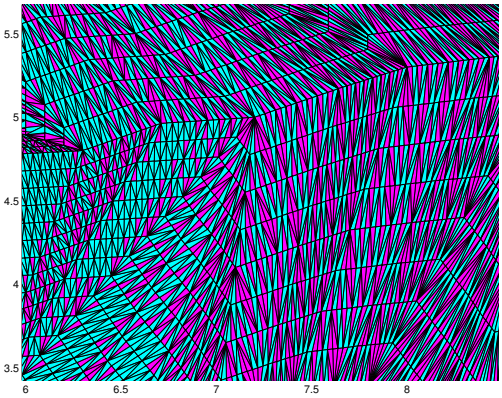
Figura 4.28: Malla pentagonal. Refinamiento global 4T-LE. Malla inicial y nivel de refinamiento uno



(a) Nivel de refinamiento dos, 1088 triángulos terminales (color azul), 2000 triángulos totales



(b) Nivel de refinamiento cuatro, 20724 triángulos terminales (color azul), 32000 triángulos totales. Detalle interior



(c) Nivel de refinamiento cuatro, 20724 triángulos terminales (color azul), 32000 triángulos totales. Detalle exterior

Figura 4.29: Malla pentagonal. Refinamiento global 4T-LE. Nivel de refinamiento dos y cuatro

Cuadro 4.7: Malla Delaunay. Medida de ángulos

N. triángulos	120	480	1920	7680	30720
Media Max.	72.9171120	90	90	90	90
Media Min.	48.9199	41.3938	41.3938	41.3938	41.3938
Máximo	100.1220	118.9241	118.924	118.924	118.924
Mínimo	36.8056	27.7368	27.7368	27.7368	27.7368
Mayores 130	0	0	0	0	0
Menores 20	0	0	0	0	0

Cuadro 4.8: Malla Pentagonal. Medida de ángulos

N. triángulos	125	500	2000	8000	32000
Media Max.	120.4169	116.5145	112.0242	108.2015	105.0403
Media Min.	9.1871	10.0152	11.0651	11.8706	12.4702
Máximo	165.8122	165.8122	165.8122	165.8122	165.8122
Mínimo	2.28	2.2800	2.2800	2.2800	2.2800
Mayores 130 (%)	25.6	16	10.2	6.6	4.35
Menores 20 (%)	95.2	91.2	87.6	84.6	82.35

Capítulo 5

Aplicaciones

5.1. Introducción

En la sección de aplicaciones presentamos tres tipos de aplicaciones específicas que encontramos al usar los algoritmos de refinamiento y desrefinamiento 2D-SBR y 2D-SBD estudiados e implementados en esta tesis.

La primera aplicación considera el uso del algoritmo de desrefinamiento en el área de modelos digitales del terreno. Concretamente se aplica a la aproximación de terrenos en base a un criterio de error. Se toman como ejemplos terrenos que corresponden a diversas zonas de la isla de Gran Canaria.

La segunda aplicación trata del modelo de niveles de detalle en visualización. Los algoritmos de simplificación o desrefinamiento permiten simplificar la complejidad de superficies mediante la reducción del número de nodos y triángulos según un parámetro de error definido. Se introducen ejemplos de niveles de detalle en el lenguaje estandar de realidad virtual, $VRML$, como una alternativa en la generación de niveles de detalle.

Por su parte, el algoritmo de refinamiento 2D-SBR es usado en la tercera aplicación como herramienta potente y versátil en un software comercial de elementos finitos, denominado *PDE Toolbox*, al que se ha incorporado. Concretamente, se usa el algoritmo 2D-SBR en la generación adaptativa de mallas para la solución de un problema no lineal de tipo elíptico.

5.2. Modelos Digitales del Terreno

5.2.1. Introducción

Se sigue en esta sección conceptos y definiciones de Leila de Floriani *et al.* en [17].

Un *sistema de información geográfica* o SIG es un sistema que incluye hardware, software y un conjunto de procedimientos orientados a la captura, procesamiento, análisis, modelado y visualización de datos espaciales. En particular, el modelado de terrenos es un área de relevancia en los SIG's que hace uso de técnicas y algoritmos de generación de mallas. En esta sección se presenta la aplicación de los algoritmos 2D-SBR y 2D-SBD al modelado del terrenos.

Los datos del terreno se relacionan con la configuración 3D de la superficie de la Tierra. La geometría del terreno es modelada mediante una superficie $2\frac{1}{2}D$. Una *superficie topográfica* o *terreno* es la gráfica de una función f de dos variables en un dominio compacto y conectado Ω en el plano euclídeo. Por ejemplo, la expresión σ en la ecuación (5.1) es la gráfica de la función f de dos variables que representa un terreno.

$$\sigma = \{(x, y, f(x, y)),\ (x, y) \in \Omega\} \tag{5.1}$$

Dado un número real q, el conjunto de las *las curvas de nivel* C_σ de un terreno representado por una función σ se define como:

$$C_\sigma(q) = \{(x, y) \in \Omega,\ f(x, y) = q\} \tag{5.2}$$

Un *Modelo Digital del Terreno* o DEM (Digital Elevation Model) es un modelo que proporciona información de un terreno tomando como referente un espacio de datos discretos. Los datos del terreno son los valores de las alturas en un conjunto de puntos conocidos del terreno $V = \{v_0, v_1, ..., v_N\} \subset \Omega$ y un conjunto de segmentos $E = \{e_0, e_1, ..., e_M\}$ que tienen como puntos extremos los puntos en V. De entre las formas posibles de representación de los datos V y E, las triangulaciones son los modelos más interesantes y mejor estudiados.

Se distinguen dos formas posibles de organizar los datos V de un terreno, *malla regular* y *red de triangulación irregular* o TIN (Triangulated Irregu-

lar Networks). Una malla regular consta de una partición del dominio Ω en polígonos regulares en la que los puntos de V están regularmente espaciados. De entre las mallas regulares, la más usada es la malla regular cuadrada. Por su parte, la red de triangulación irregular consta de una serie de datos no regulares sobre el dominio Ω. Ambas organizaciones de los datos han sido estudiadas por diversos autores. Algoritmos para mallas regulares en terrenos y sus propiedades han sido estudiados entre otros en [46, 64]. En particular, la triangulación tipo Delaunay es la más usada para la representación del terreno mediante una red de triangulación irregular [76].

Se consigue una mejor representación (precisión) del terreno cuanto más datos se disponga. Sin embargo, el procesamiento de grandes cantidades de datos conlleva un alto coste en almacenamiento y tiempo de cómputo. Por ello se recurre a técnicas que reduzcan estos costes equilibrando tamaño de representación y precisión deseada.

Un *modelo aproximado del terreno* es un modelo que se construye en base al uso de un conjunto más reducido de los datos del terreno. El error ϵ que se comete en la aproximación es medido con respecto al modelo de referencia construido con los datos disponibles del terreno: una elección muy común para ϵ es la máxima diferencia entre la altura en un punto de los datos y la altura interpolada en el modelo aproximado. La precisión de un modelo aproximado del terreno, la cual no tiene en cuenta otras fuentes de error, como por ejemplo el muestreo respecto de la realidad se define como:

$$P = \frac{1}{1+\epsilon} \tag{5.3}$$

Además, para medir la bondad de ajuste de una representación concreta del terreno es posible utilizar la métrica *SNR (Signal Noise Ratio)*, también usada en [66] en otra aplicación del desrefinamiento de mallas bidimensionales. El SNR, ecuación (5.4), proporciona la relación de los valores originales de la alturas en los distintos puntos del terreno frente al error cometido al eliminar nodos. Como su nombre indica representa la energía de una función respecto del ruido y se mide en decibelios:

$$SNR = 10 \log \frac{\sum_{n=1}^{N} s^2(n)}{\sum_{n=1}^{N} e^2(n)} \tag{5.4}$$

donde $s(n)$ es el valor exacto de la función que se representa y $e(n)$ el error que se comete cuando se quita un nodo en la malla. El recorrido de la función SNR es $[0, \infty)$.

Así, dependiendo del problema y de las necesidades, se puede optimizar el rendimiento de una aplicación si se adopta un modelo aproximado del terreno en lugar del conjunto completo de datos disponibles. La *generalización de terrenos* trata sobre cómo alcanzar el ratio óptimo entre precisión y tamaño de representación para un terreno. En este sentido se consideran dos criterios de optimización diferentes:

1. Minimizar el número de vértices del modelo para una precisión dada y
2. Maximizar la precisión para un número dado de vértices.

Las técnicas de refinamiento y desrefinamiento (también llamada simplificación), son usadas para alcanzar los criterios anteriores en generalización de terrenos [17] y en generación y simplificación de escenas en gráficos por ordenador [15].

En el siguiente apartado se aplican los algoritmos 2D-SBR y 2D-SBD y triangulaciones basadas en la partición 4T-LE para minimizar el número de vértices de un modelo del terrreno considerando diferentes valores de precisión.

5.2.2. Generalización de terrenos

A continuación se presentan dos ejemplos de aplicación de los algoritmos 2D-SBR y 2D-SBD a la generalización de terrenos. Los datos de los terrenos fueron obtenidos con una malla regular de paso variable (en función del terreno) por procesos fotogramétricos. Los datos de las alturas se obtienen por curvas de nivel cada 5 metros de altimetría. Las zonas estudiadas corresponden a dos zonas del norte de la isla de Gran Canaria.

Los experimentos presentados fueron calculados en un procesador AMD-K7 a 1100 Mhz con 384 Mb. de Ram.

El primer ejemplo considera las inmediaciones del puerto de Agaete en el norte de Gran Canaria con un paso de 15 metros en dirección x y 15 metros

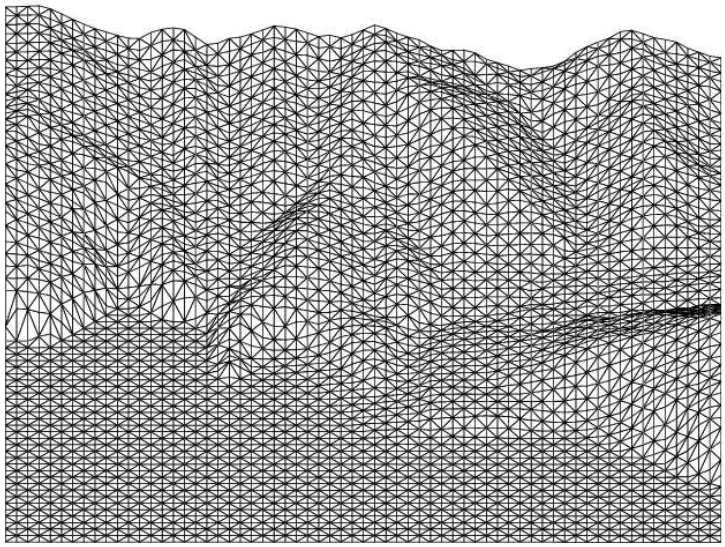
en dirección y. El terreno se extiende sobre un área de 990 metros cuadrados y con altitud máxima de 253 metros sobre el nivel del mar. Se presentan en las figuras 5.1 y 5.2 los datos iniciales disponibles del terreno, en una malla de 4225 nodos y 8192 triángulos. En las figuras siguientes, desde la 5.3 a la 5.10 se presentan los resultados de los terrenos aproximados, considerando desrefinamiento y usando como valor de error 1, 6, 15 y 36 metros respectivamente. Para cada caso se proporcionan las mallas 2D y las curvas de nivel, lo que da una idea visual de la eliminación de nodos en cada caso y del correspondiente terreno según las curvas de nivel. El renderizado del terreno resultante en cada caso es realizado con las herramientas de visualización 3D del $Matlab^{TM}$.

En la figura 5.11 se presenta la gráfica del error en metros respecto del número de nodos de las mallas de obtenidas. En la figura 5.11 se representa gráficamente la evolución del SNR (Signal Noise Ratio) en unidades de decibelios frente al número de nodos de las mallas obtenidas en cada paso. Finalmente, en la figura 5.13 muestra la evolución del tiempo en segundos que tarda el algoritmo 2D-SBD en la obtención de las representaciones con cada error considerado.

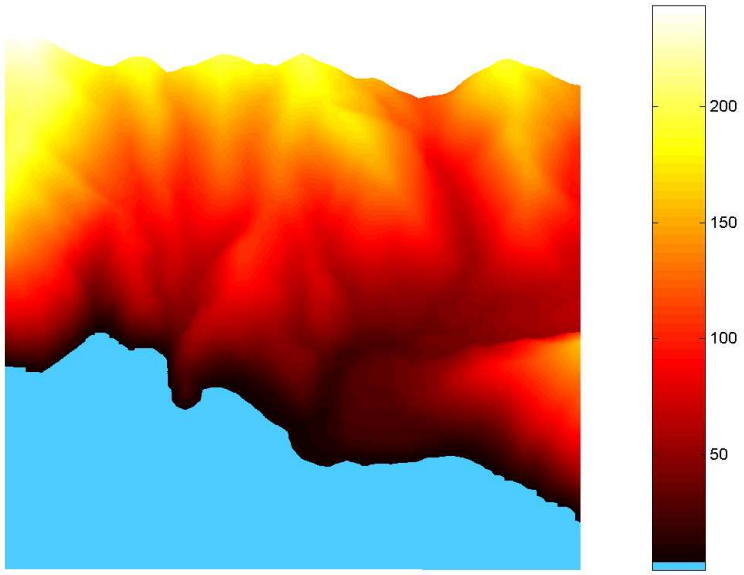
El segundo ejemplo corresponde a la costa norteña de Gran Canaria, ciudad de Gáldar. En este caso se considera un paso de 19 metros en x e y respectivamente sobre una extensión cuadrada de 4883 metros con una altitud máxima de 445 metros sobre el nivel del mar. Se presenta en la figura 5.14 los datos del terreno inicial, en una malla de 66049 nodos y 131072 triángulos. En las figuras siguientes, desde la 5.15 a la 5.22, se presentan los resultados de los terrenos aproximados, considerando desrefinamiento y usando como valor de error 10, 15, 40 y 90 metros respectivamente. Para cada caso, se proporcionan las mallas 2D y las curvas de nivel, lo que da una idea visual de la eliminación de nodos y del correspondiente terreno según las curvas de nivel en cada caso. El renderizado del terreno resultante en cada caso es realizado con las herramientas de visualización 3D del $Matlab^{TM}$. En la tabla 5.1 de la página 190 se muestran los tiempos de renderizado del terreno para determinados casos según el error y además los tamaños de los ficheros resultantes al almacenar dichos terrenos.

En la figura 5.23 página 185 se presenta la gráfica del error en metros

respecto del número de nodos de las mallas de obtenidas. En la figura 5.24 se representa gráficamente la evolución del SNR (Signal Noise Ratio) en unidades de decibelios frente al número de nodos de las mallas obtenidas en cada paso. Finalmente para el ejemplo de Agaete, en la figura 5.25 página 185 se presenta la evolución del tiempo en segundos que tarda el algoritmo 2D-SBD en la obtención de las representaciones con cada error considerado.

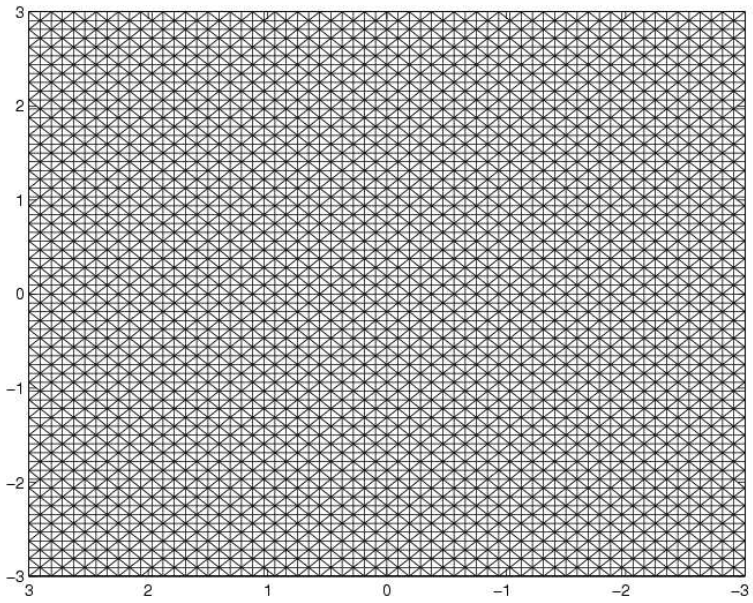


(a) Modelo inicial de la costa de Agaete, 4225 nodos, 8192 triángulos

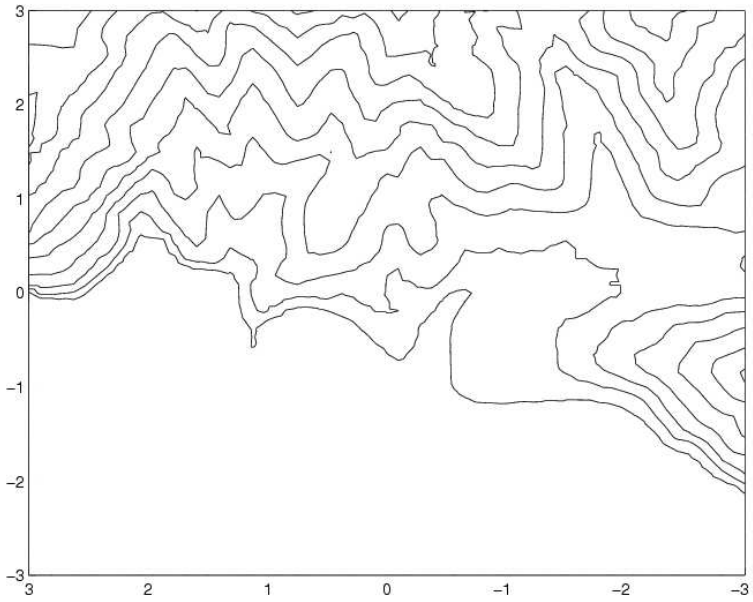


(b) Imagen renderizada del modelo inicial

Figura 5.1: Representación inicial del terreno de Agaete

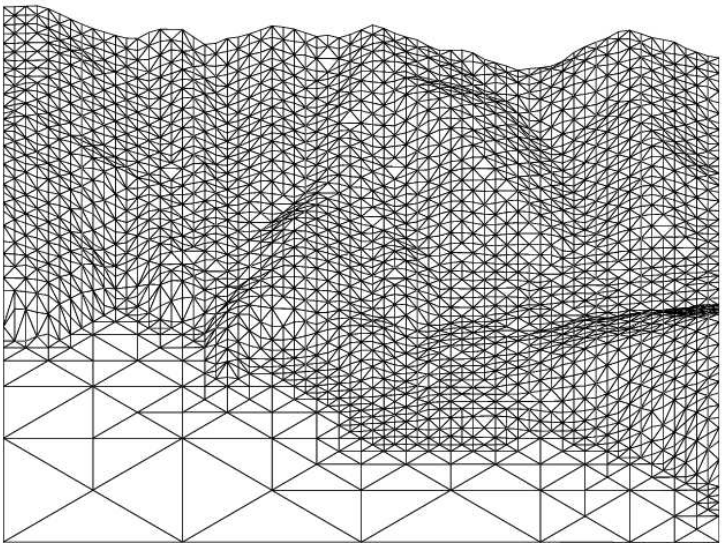


(a) Malla 2D inicial

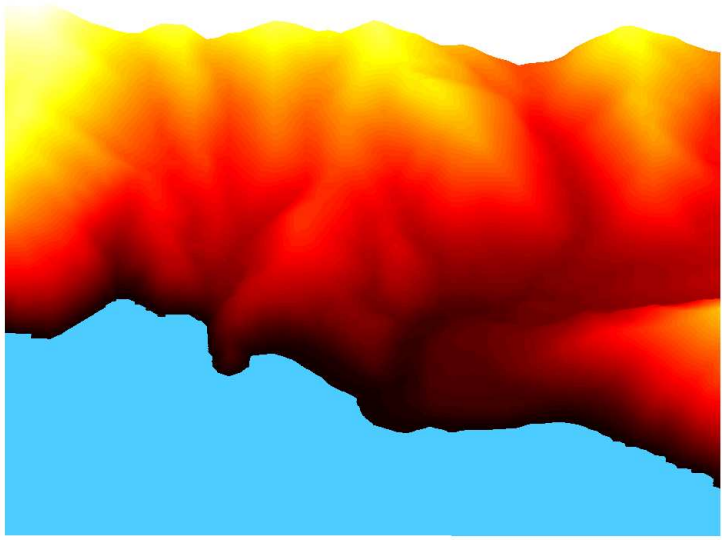


(b) Curvas de nivel

Figura 5.2: Malla 2D y curvas de nivel del terreno de Agaete

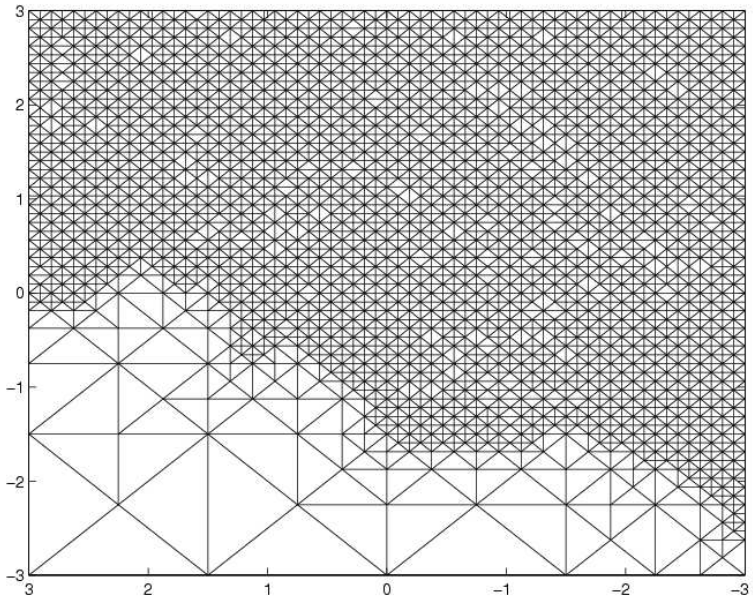


(a) Aproximación con 1 metro de error, 2837 nodos, 5505 triángulos, SNR=45.0852

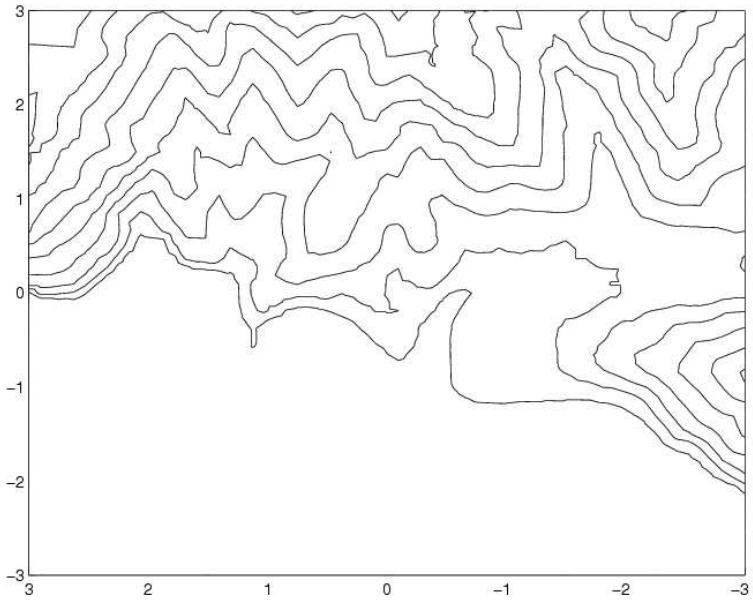


(b) Imagen renderizada

Figura 5.3: Representación con error de 1 metro del terreno de Agaete

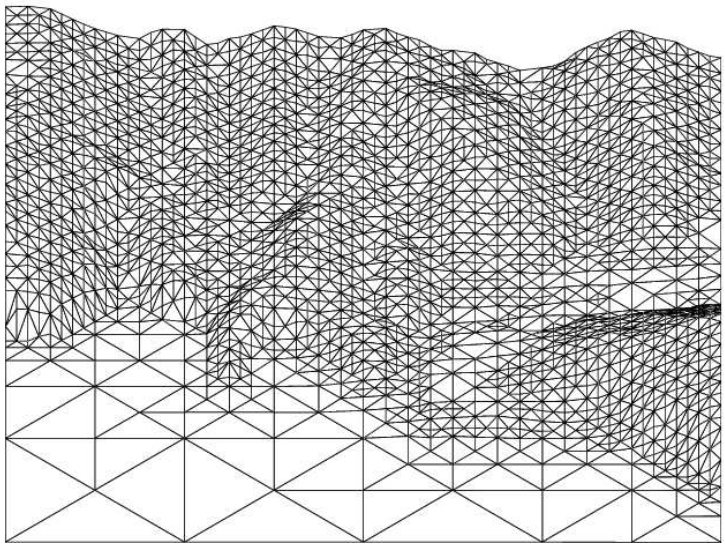


(a) Malla 2D

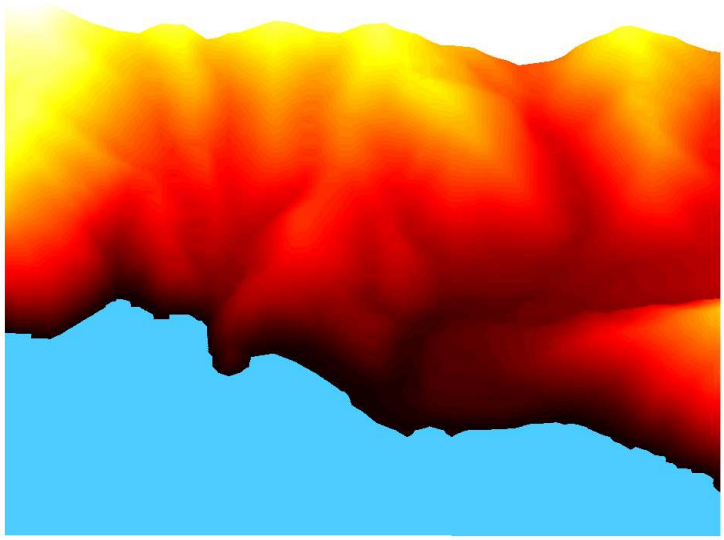


(b) Curvas de nivel

Figura 5.4: Malla 2D y curvas de nivel con error de 1 metro del terreno de Agaete

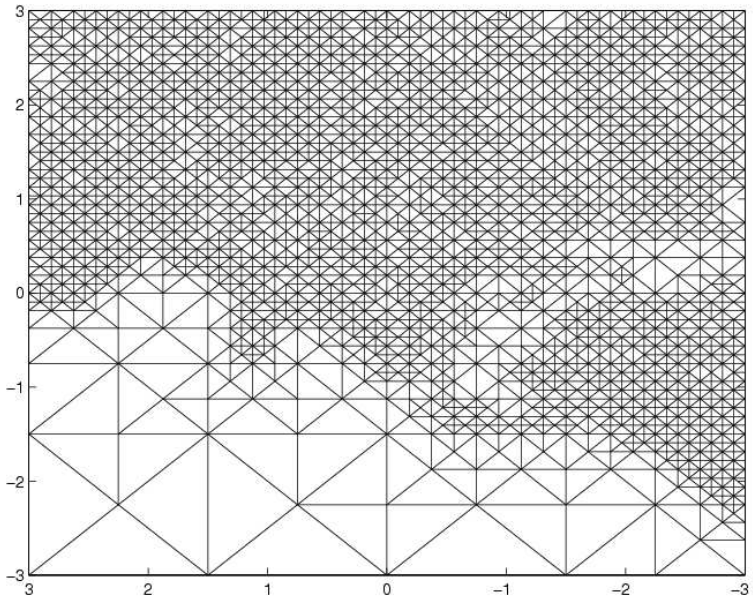


(a) Aproximación con 6 metros de error, 2344 nodos, 4547 triángulos, SNR=37.5181

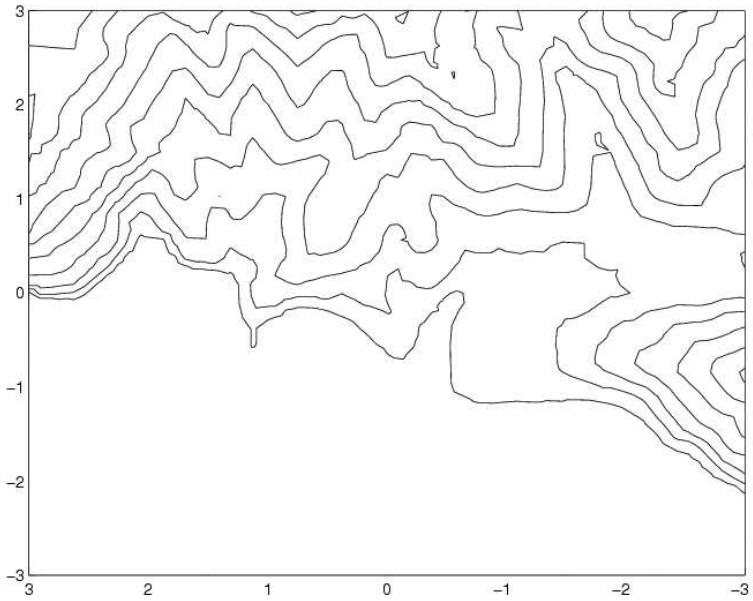


(b) Imagen renderizada

Figura 5.5: Representación con error de 6 metros del terreno de Agaete

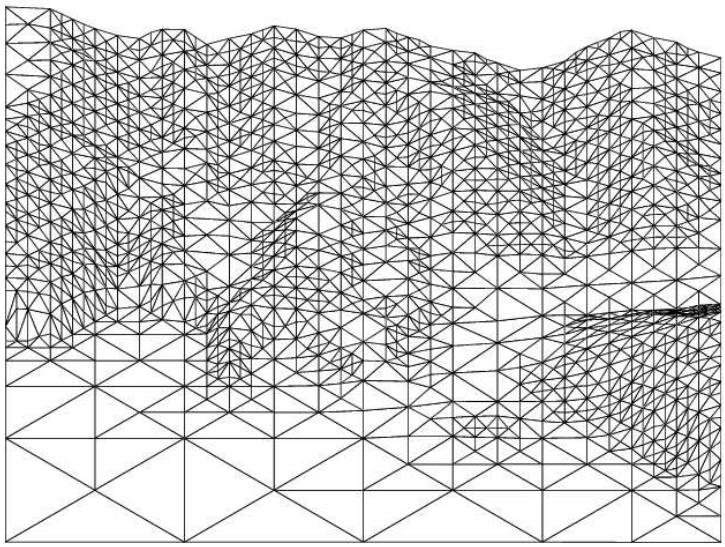


(a) Malla 2D

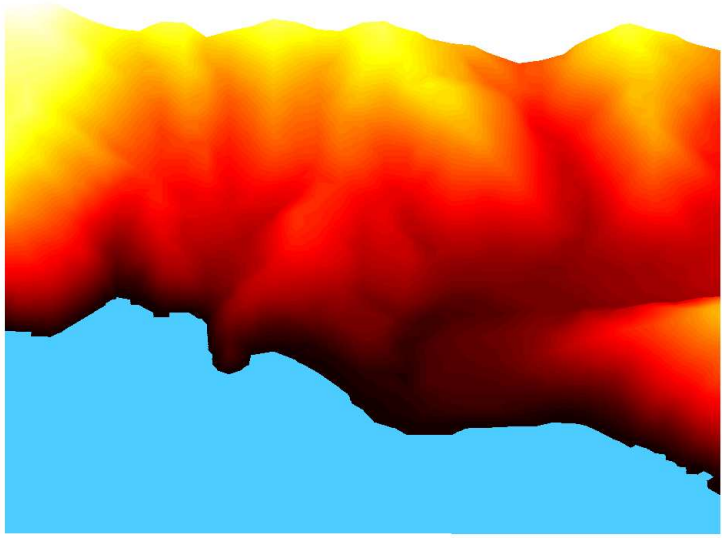


(b) Curvas de nivel

Figura 5.6: Malla 2D y curvas de nivel con error de 6 metros del terreno de Agaete

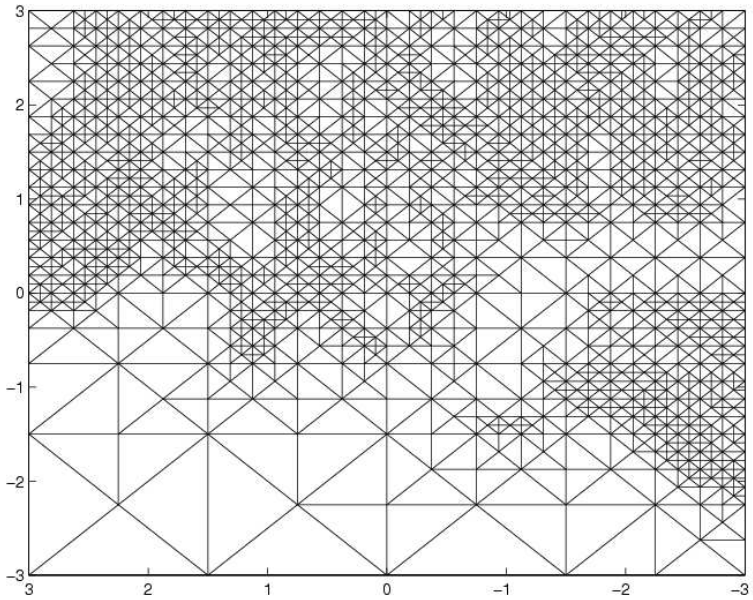


(a) Aproximación con 15 metros de error, 1588 nodos, 3063 triángulos, SNR=32.6170

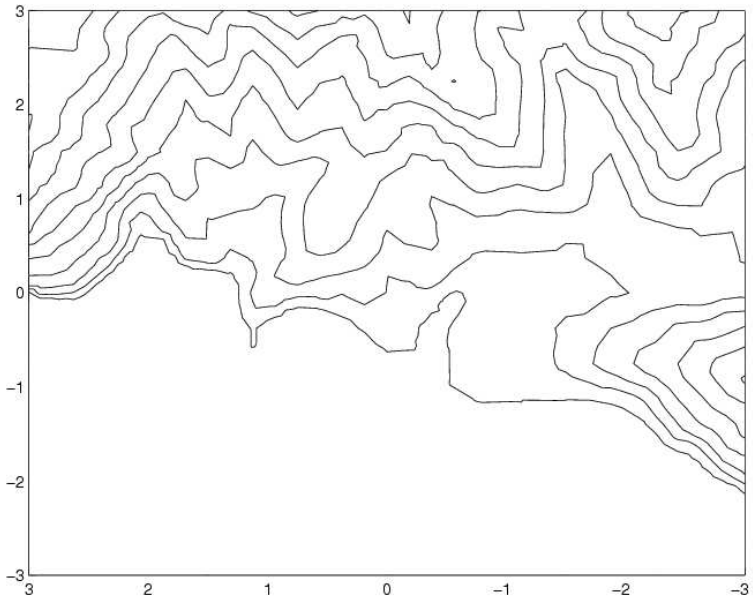


(b) Imagen renderizada

Figura 5.7: Representación con error de 15 metros del terreno de Agaete

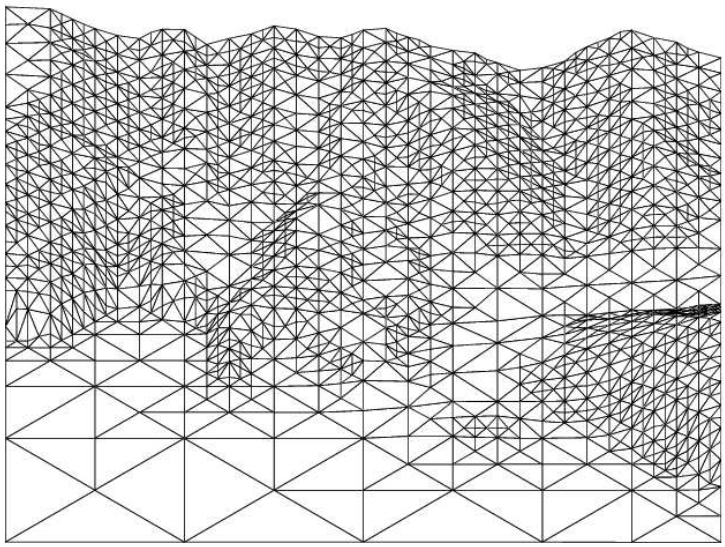


(a) Malla 2D

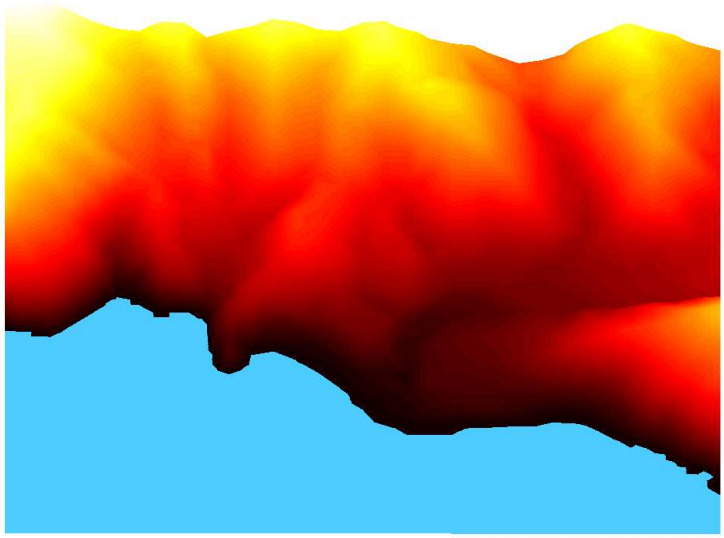


(b) Curvas de nivel

Figura 5.8: Malla 2D y curvas de nivel con error de 15 metros del terreno de Agaete

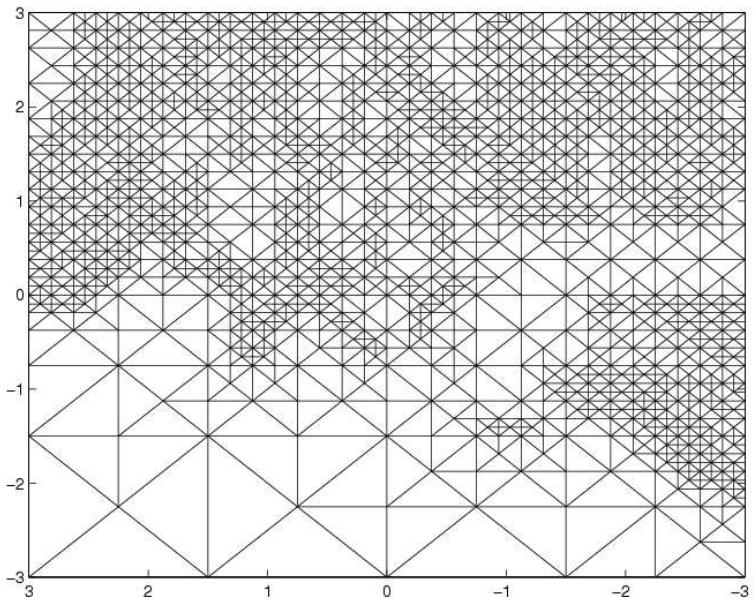


(a) Aproximación con 36 metros de error, 416 nodos, 774 triángulos, SNR=26.1788

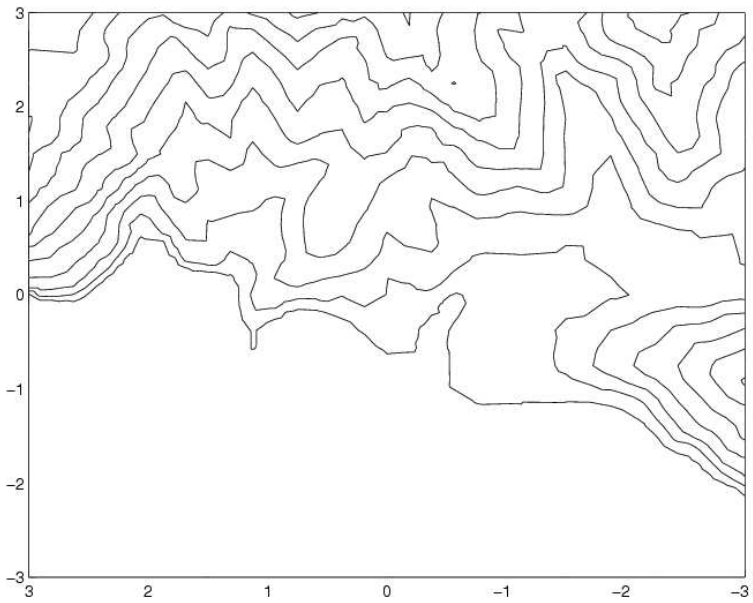


(b) Imagen renderizada

Figura 5.9: Representación con error de 36 metros del terreno de Agaete



(a) Malla 2D



(b) Curvas de nivel del terreno

Figura 5.10: Malla 2D y curvas de nivel con error de 36 metros del terreno de Agaete

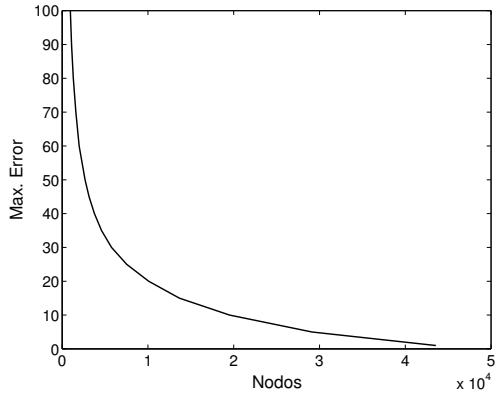


Figura 5.11: Gráfica del error en metros frente al número de nodos

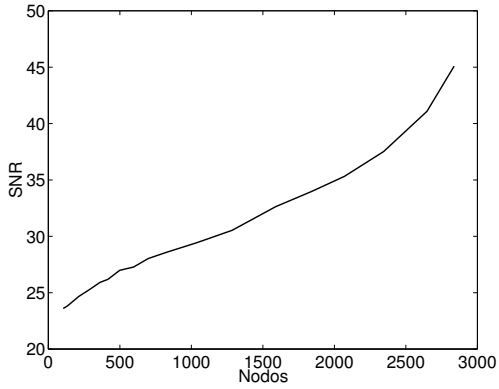


Figura 5.12: Gráfica del SNR en decibelios frente al número de nodos

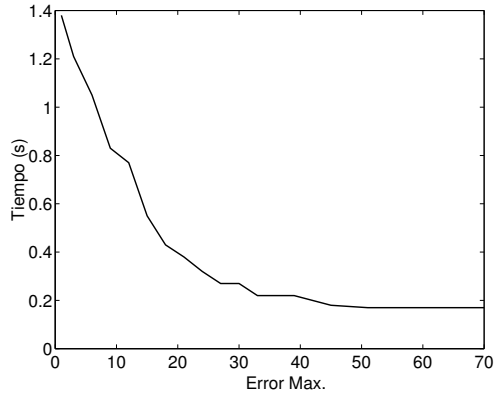
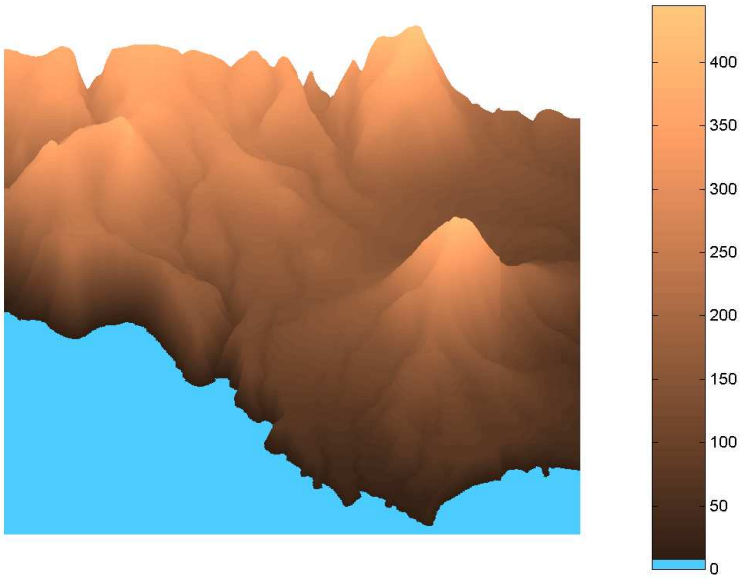
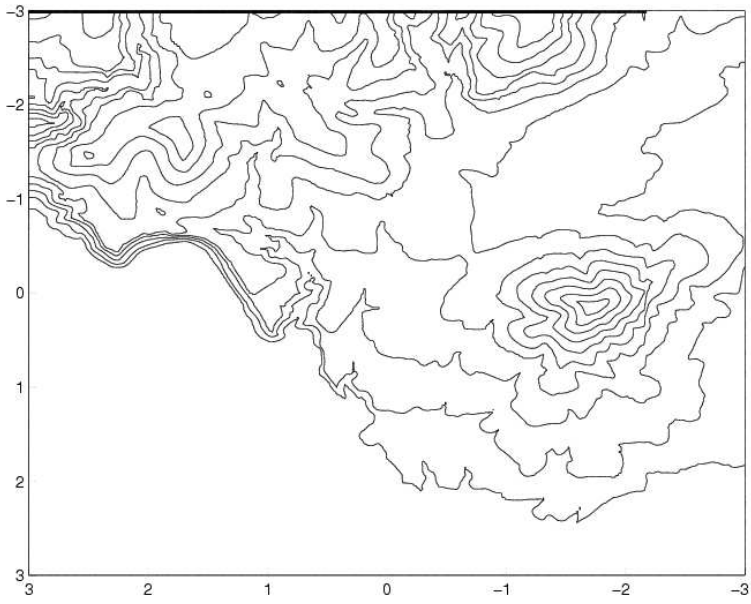


Figura 5.13: Tiempo de obtención de las mallas frente al nivel de error

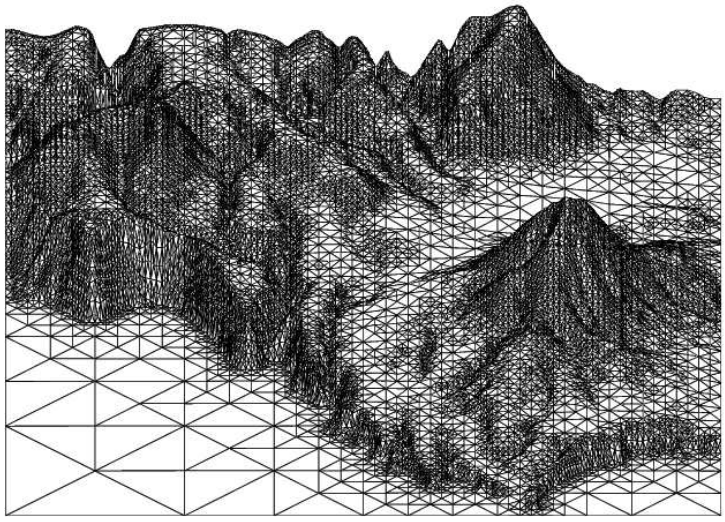


(a) Terreno inicial de la costa de Gáldar, 66049 nodos, 131072 triángulos

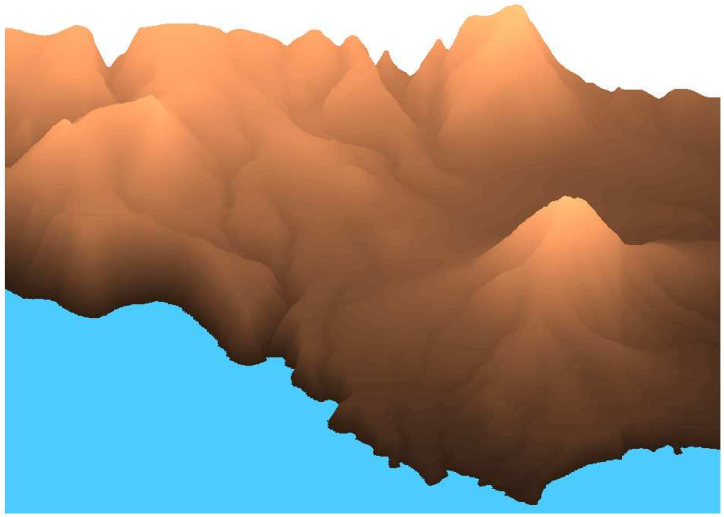


(b) Curvas de nivel

Figura 5.14: Representación inicial del terreno de Gáldar

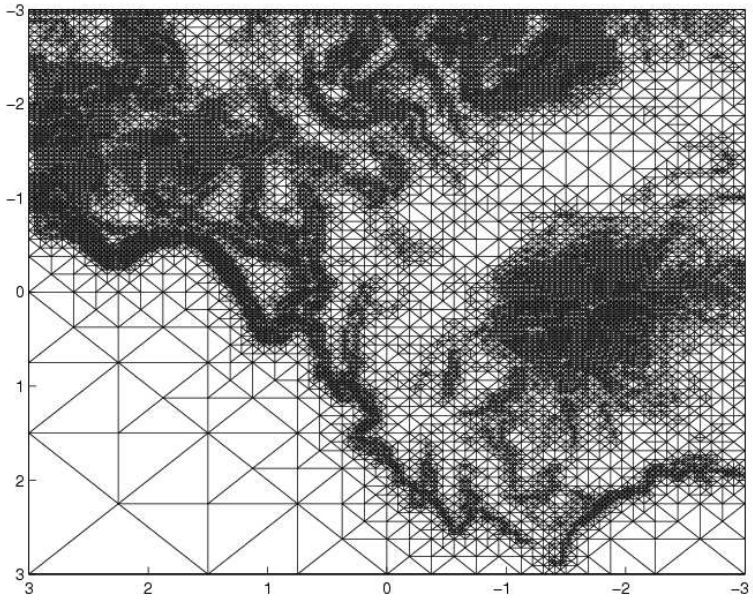


(a) Aproximación con 10 metros de error, 19506 nodos, 38732 triángulos, SNR=43.2667

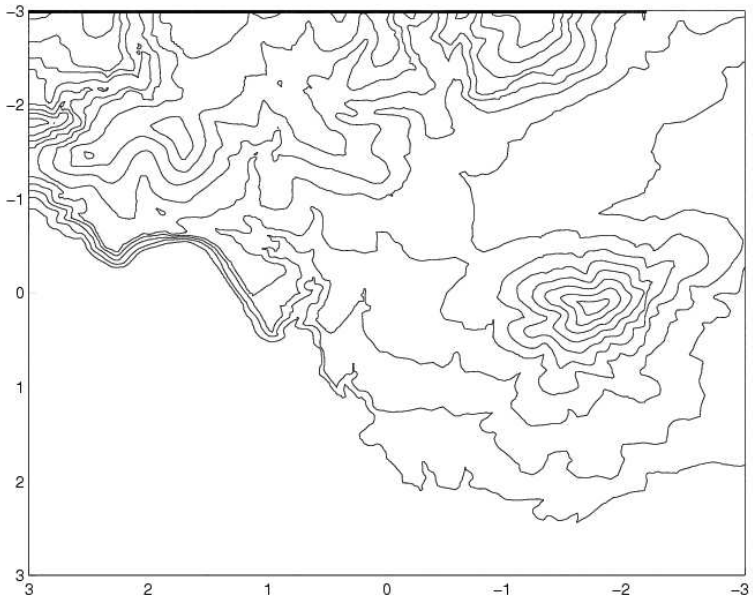


(b) Imagen renderizada

Figura 5.15: Representación con error de 10 metros del terreno de Gáldar

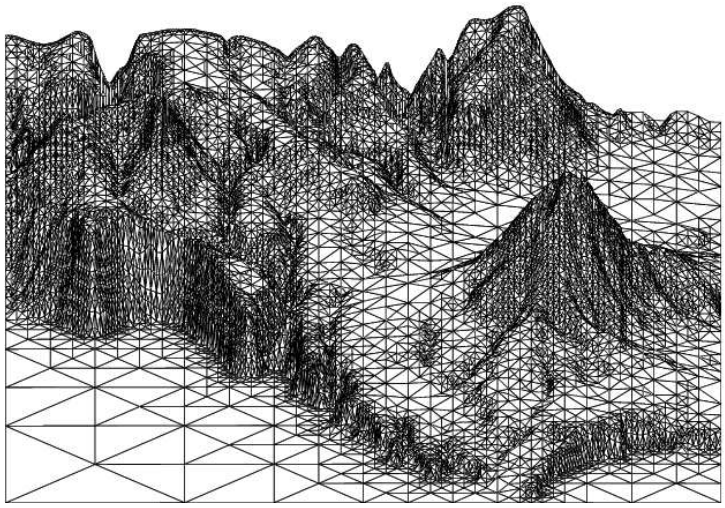


(a) Malla 2D

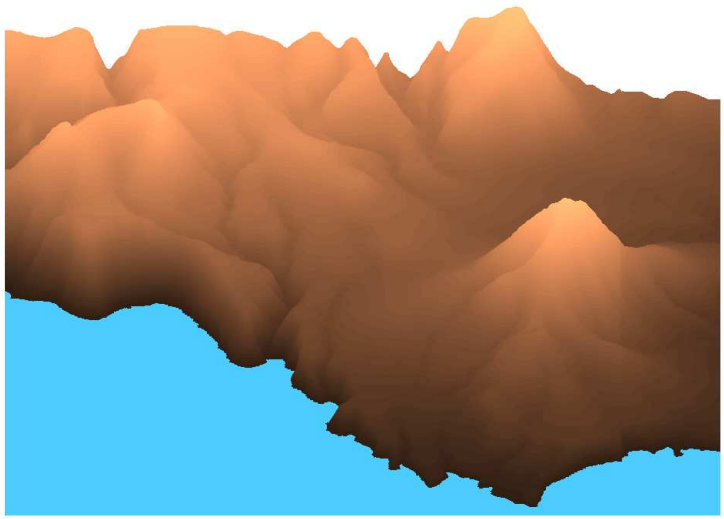


(b) Curvas de nivel

Figura 5.16: Malla 2D y curvas de nivel con error de 10 metros del terreno de Gáldar

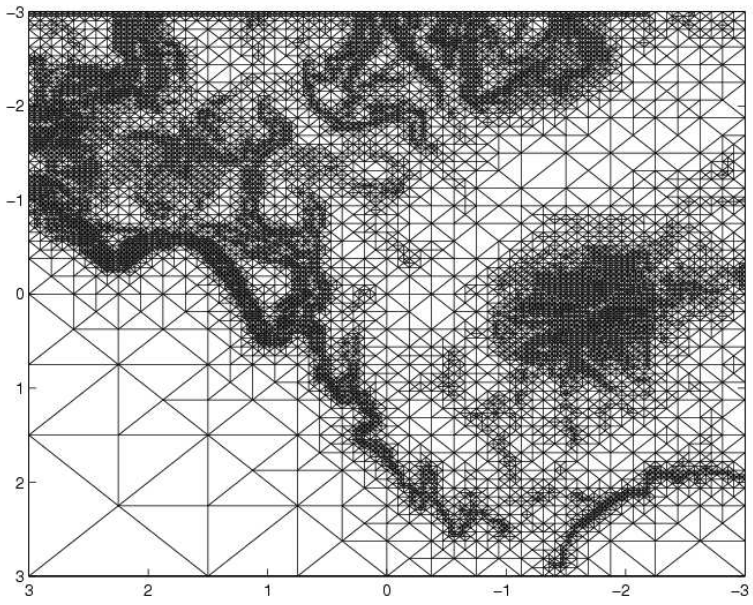


(a) Aproximación con 15 metros de error, 13661 nodos, 27073 triángulos, SNR=33.0100

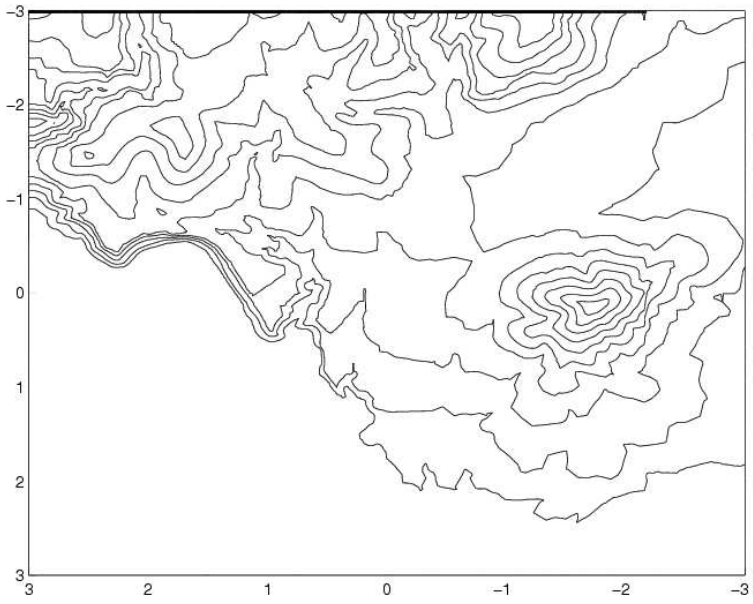


(b) Imagen renderizada

Figura 5.17: Representación con error de 15 metros del terreno de Gáldar

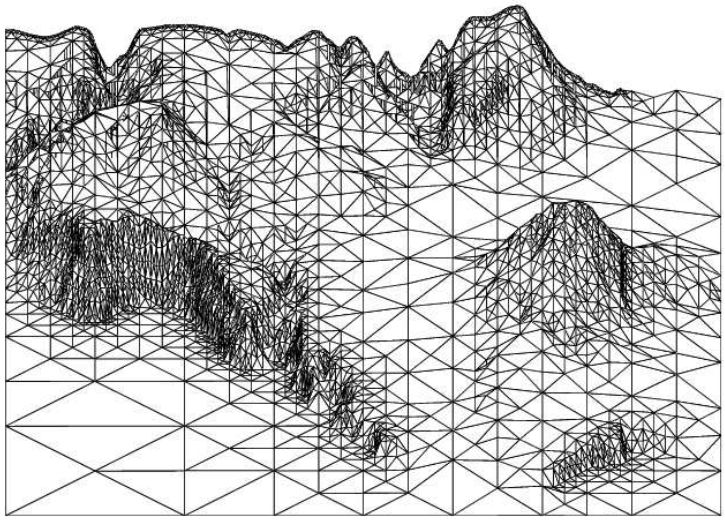


(a) Malla 2D

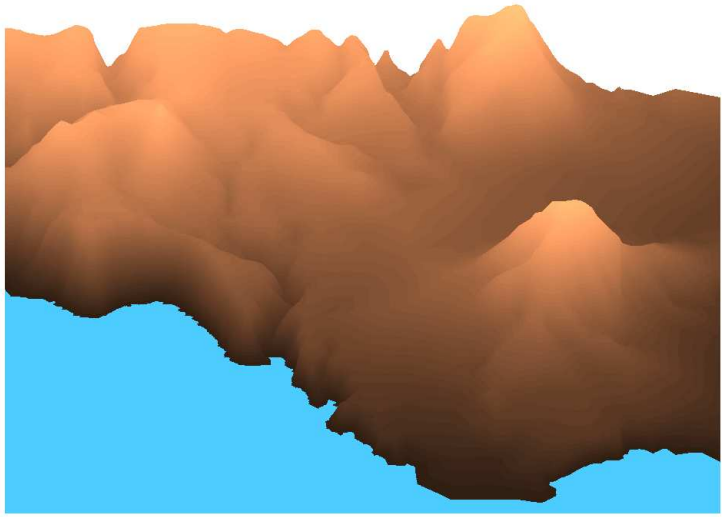


(b) Curvas de nivel

Figura 5.18: Malla 2D y curvas de nivel con error de 15 metros del terreno de Gáldar

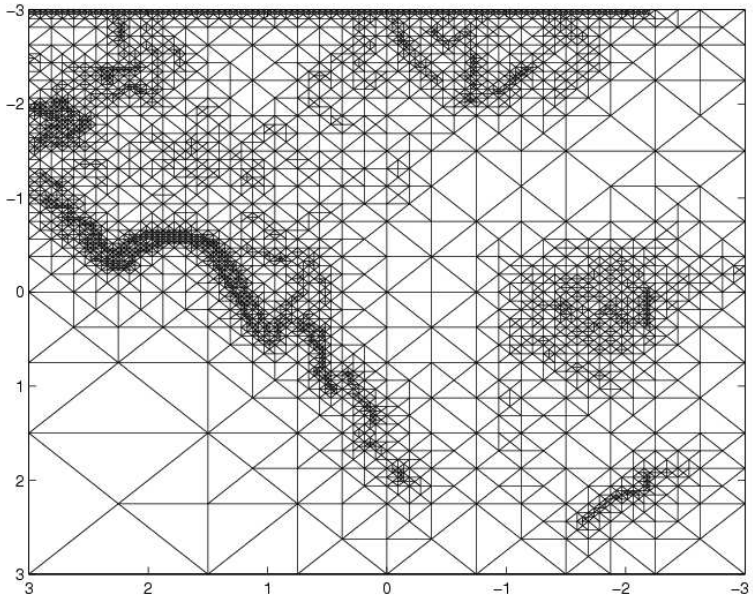


(a) Aproximación con 40 metros de error, 3755 nodos, 7333 triángulos, SNR=34.4369

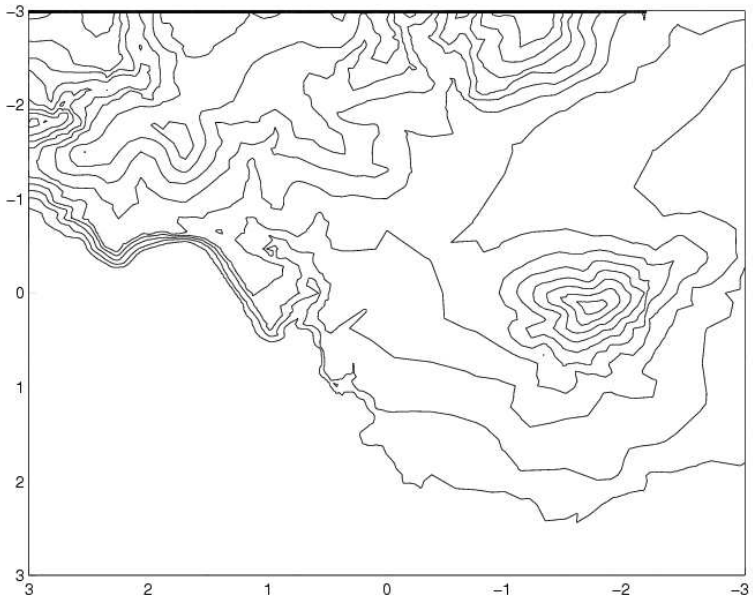


(b) Imagen renderizada

Figura 5.19: Representación con error de 40 metros del terreno de Gáldar

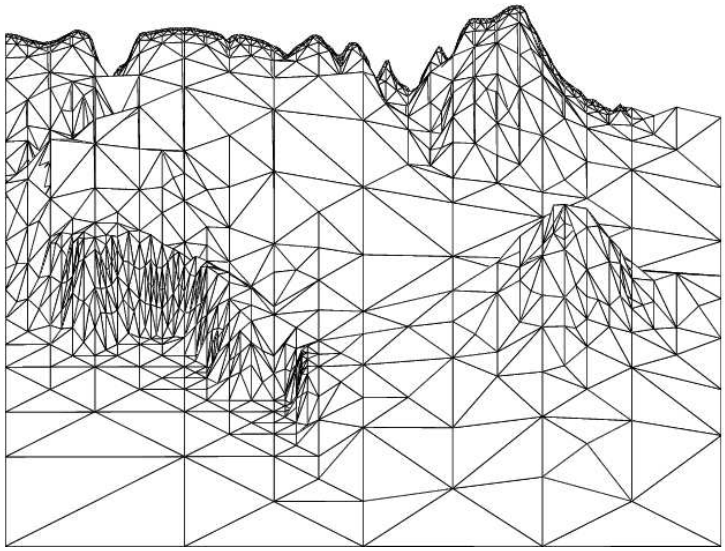


(a) Malla 2D

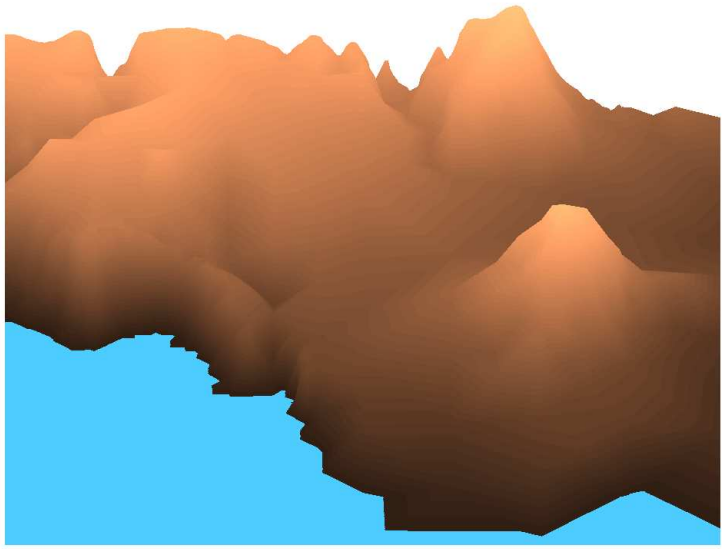


(b) Curvas de nivel

Figura 5.20: Malla 2D y curvas de nivel con error de 40 metros del terreno de Gáldar

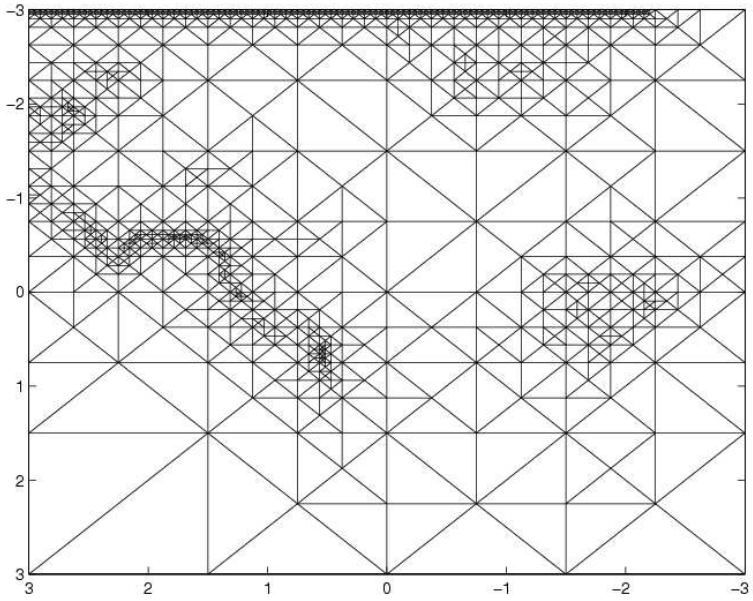


(a) Aproximación con 90 metros de error, 1094 nodos, 2035 triángulos, SNR=31.0967

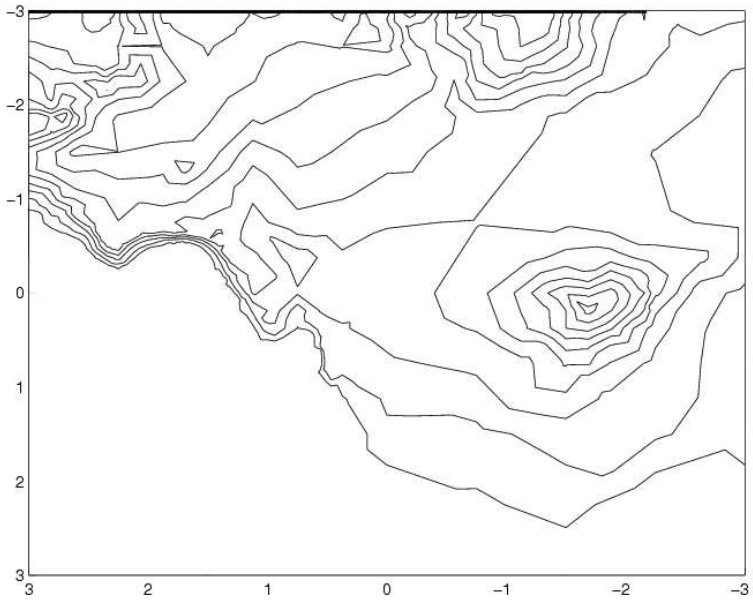


(b) Imagen renderizada

Figura 5.21: Representación con error de 90 metros del terreno de Gáldar



(a) Malla 2D



(b) Curvas de nivel

Figura 5.22: Malla 2D y curvas de nivel con error de 90 metros del terreno de Gáldar

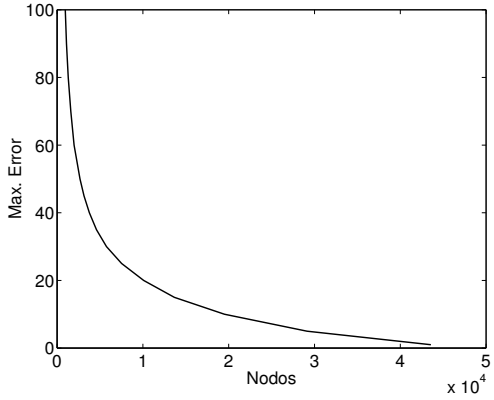


Figura 5.23: Gráfica del error en metros frente al número de nodos

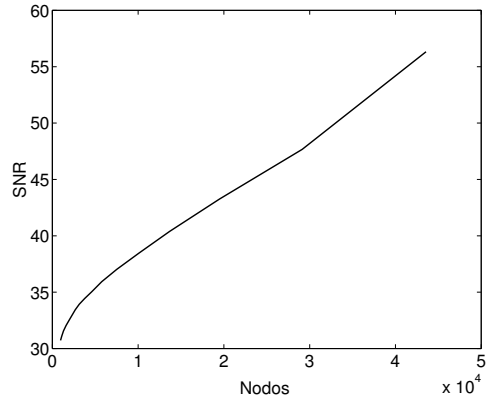


Figura 5.24: Gráfica del SNR en decibelios frente al número de nodos

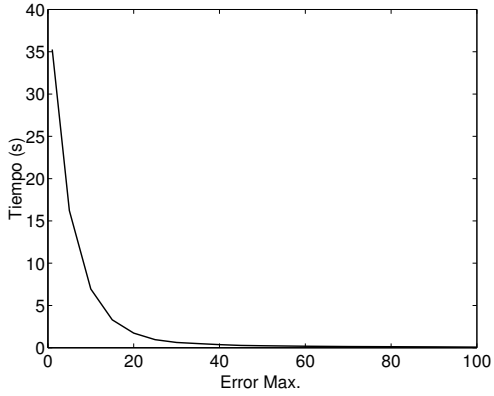


Figura 5.25: Tiempo de obtención de las mallas frente al nivel de error

5.3. Niveles de Detalle

La visualización interactiva de gráficos 3D y superficies es un aspecto crucial en los modernos sistemas de gráficos por ordenador. Uno de los problemas en la visualización en tiempo real es el renderizado de las superficies 3D puesto que el tiempo que consume dicha operación es elevado cuando la magnitud del gráfico (en término de nodos o polígonos) es de gran tamaño. Para mejorar el rendimiento en las operaciones de *rendering* la complejidad de los gráficos debe simplificarse bastante sin que ello suponga pérdida notable en la precisión y percepción visual de los gráficos. Un modelo versátil para lograr eficiencia, no sólo del rendering de una escena 3D sino de la manipulación de los gráficos es el modelo de *Niveles de Detalle* o *LOD (Levels Of Detail).*

Un modelo de niveles de detalle consiste en una serie de representaciones aproximadas de un gráfico 3D o superficie, ver Leila de Floriani *et al.* en [17]. Así un modelo basado en niveles de detalle permite que el renderizado de una escena en 3D admita diferentes resultados, más o menos costosos, dependiendo de las distintas aproximaciones almacenadas como niveles de detalle. Otra utilidad reciente de los niveles de detalle es que, por ejemplo, si se dispone de un sistema de realidad virtual con la capacidad de navegar y sumergirse en una escena 3D, los niveles de detalle permiten de forma dinámica y en tiempo real el acercamiento o alejamiento de los objetos. Se usan entonces distintas aproximaciones con niveles de detalle variable en función de la cercanía o alejamiento a los objetos. El modelo de niveles de detalle es cada vez más utilizado en lenguajes y paquetes de software, como por ejemplo en $OpenInventor^{TM}$ [98] y $VRML$ [35].

Un sistema que genere niveles de detalle debe reunir entre otras, las siguientes características [46].

1. Simplificacion de la geometría. Se trata de conseguir una reducción de los polígonos (triángulos en nuestro ámbito) que forman la geometría.
2. Cambios continuos y suaves entre distintos niveles de detalle.
3. Generación dinámica de niveles de detalle en tiempo real. El tiempo en la generación de los distintos niveles de detalle ha ser pequeño, no

implicando el deterioro en la rapidez de la aplicación.

4. Posibilidad de elección del nivel de detalle. El usuario debe especificar mediante alguna métrica o error, la precisión que debe tener cada nivel de detalle.

El tratamiento y la aplicación que haremos en este trabajo considera superficies que son triangulaciones según la partición 4T-LE aunque las ideas son análogas para gráficos 3D en general.

Para calcular las distintas representaciones de niveles de detalle pueden usarse distintos enfoques en la selección de los puntos relevantes en cada nivel de detalle. El enfoque más común consiste en el *sub–muestreo* o *super–muestreo* de los datos originales para generar la triangulación del nivel de detalle. La desventaja que tiene este enfoque es que la adaptividad de los niveles de detalle a la superficie original no es la mejor, ya que la selección de los puntos sólo depende de su posición en la malla y no de características de la superficie, como regiones con gradiente pronunciado, suavizado de zonas o una medida de error que actúe de control. La ventaja que tiene este enfoque es su simplicidad y rapidez en la generación de los niveles de detalle.

Otro enfoque posible es usar adaptividad en la generación de las distintas mallas que forman los niveles de detalle. La adaptividad es controlada por una medida del error, por ejemplo el máximo error en el valor $f(x, y)$ (altura si se trata de terrenos). Es decir, los puntos de la malla seleccionados para cada nivel de detalle se eligen en base a un criterio de desrefinamiento, ver página 130 del capítulo 4. Este enfoque coincide con un problema de desrefinamiento en 2D, igual al planteado en la sección 1 de este capítulo en generalización de terrenos. La ventaja que tiene el enfoque del desrefinamiento para lograr adaptividad es que la calidad de las representaciones de los distintos niveles de detalle es mayor que usando muestreo y además la calidad es controlada por una medida de error dada por el usuario. Otra ventaja notable es que se reduce considerablemente el espacio de almacenamiento para cada nivel de detalle y esto es crítico cuando se manejan varios cientos de miles o millones de datos. La desventaja que tienen este enfoque es que consume más tiempo en la generación de los niveles de detalle respecto del muestreo.

Si un modelo de niveles de detalle considera técnicas de simplificación o desrefinamiento, como el explicado en el enfoque de la adaptividad, se denomina modelo Multirresolución, ver Leila de Floriani *et al.* [17].

La figura 5.26 representa esquemáticamente los distintos puntos de vista que se asocian a los distintos niveles de detalle en la visualización de una superficie.

El objetivo en esta sección es aplicar el algoritmo de desrefinamiento 2D-SBD en la generación de niveles de detalle e incorporar el resultado en código $VRML$. Para visualizar el código $VRML$ recurrimos al programa gratuito $Cosmoplayer^{TM}$ que es uno de los más conocidos para visualizar ficheros $VRML$.

Para ilustrar el ejemplo tomamos la superficie dada por la ecuación siguiente:

$$z = \cos(x \cdot y) \tag{5.5}$$

definida en el dominio $x \in [-3, 3]$, $y \in [-3, 3]$ y $z \in [-1, 1]$.

Para representar la superficie tomamos una malla regular de 65×65 puntos. Se utiliza el algoritmo de desrefinamiento 2D-SBD para obtener tres niveles de detalle correspondiendo a los valores de error máximo 10 %, 30 %, y 60 %. La conversión a $VRML$ de los datos de la malla que se generan, así como de otros detalles del renderizado y presentación, se realiza mediante una rutina en código Matlab que genera código portable $VRML$ 2.0.

Como resultado se obtienen tres representaciones que llamaremos $LOD2$, $LOD3$, $LOD4$ y la representación inicial que llamaremos $LOD1$. En la figura 5.27 se presenta la malla 2D y la representación $VRML$ del nivel de detalle a máxima resolución, $LOD1$. Las figuras 5.28, 5.29, 5.30 corresponden respectivamente a los niveles de detalles obtenidos $LOD2$, $LOD3$ y $LOD4$. Los distintos niveles de detalle aparecen automáticamente cuando el usuario toma el control y se aleja de la superficie dentro del escenario $VRML$ del $Cosmoplayer$.

Finalmente se presenta otro ejemplo de generación de niveles de detalle. En

la tabla 5.1 se han calculado tres niveles de detalle (aparte de la representación dada $LOD1$) para el ejemplo del terreno de Gáldar de la sección anterior. Se ha usado el algoritmo de desrefinamiento 2D-SBD y se muestran los datos del error cometido en cada nivel de detalle, el número de triángulos y puntos de las mallas asociadas, el tiempo que se tarda en renderizar la superficie en segundos y el tamño en bytes del fichero asociado para almacenar los puntos, triángulos y la función de la altura.

Cuadro 5.1: Niveles de detalle para el terreno de Gáldar

LOD	1	2	3	4
Error (m.)	0	20	40	60
N. triángulos	131072	10096	3755	1975
N. puntos	66049	19962	73333	3789
T. rendering (s.)	1.310	0.220	0.110	0.060
Tamaño (Kb.)	3653	406	156	86

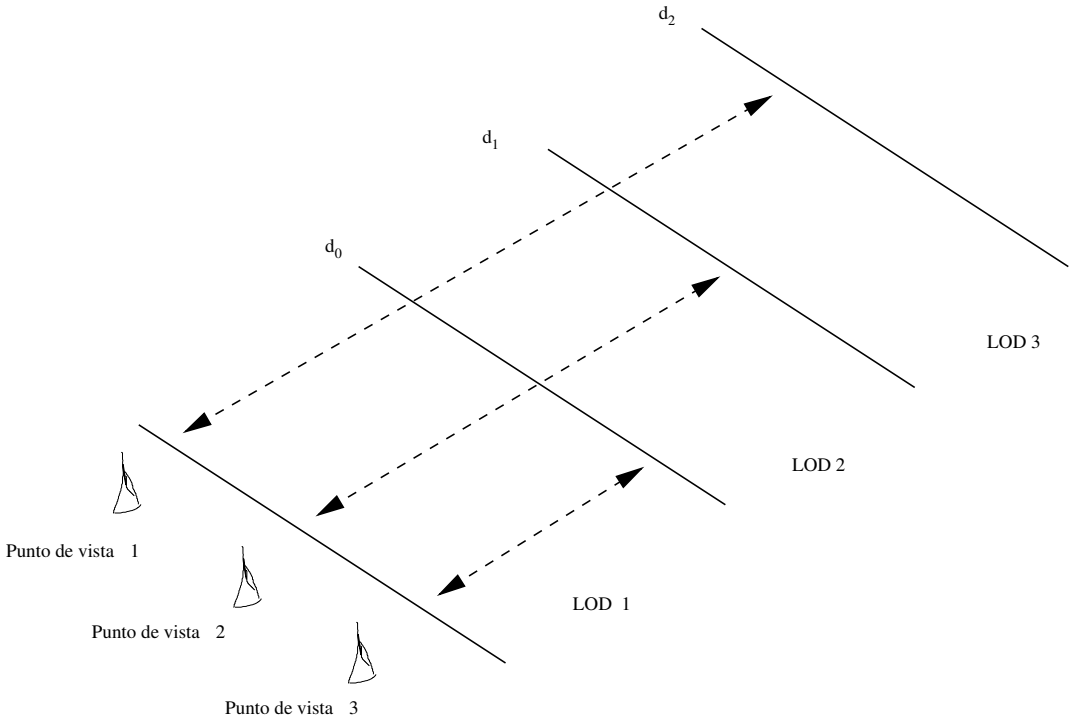
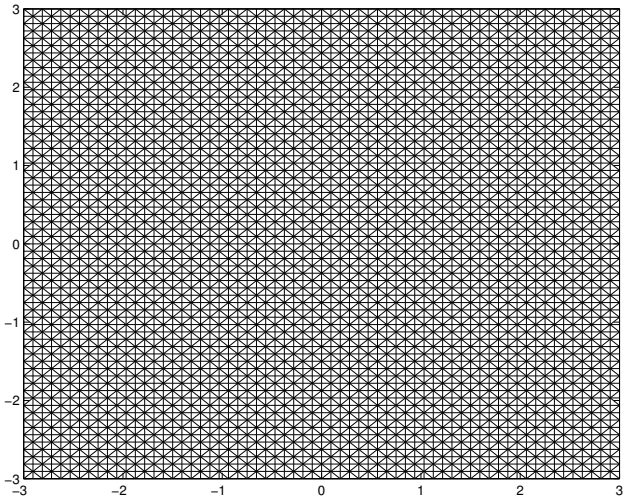
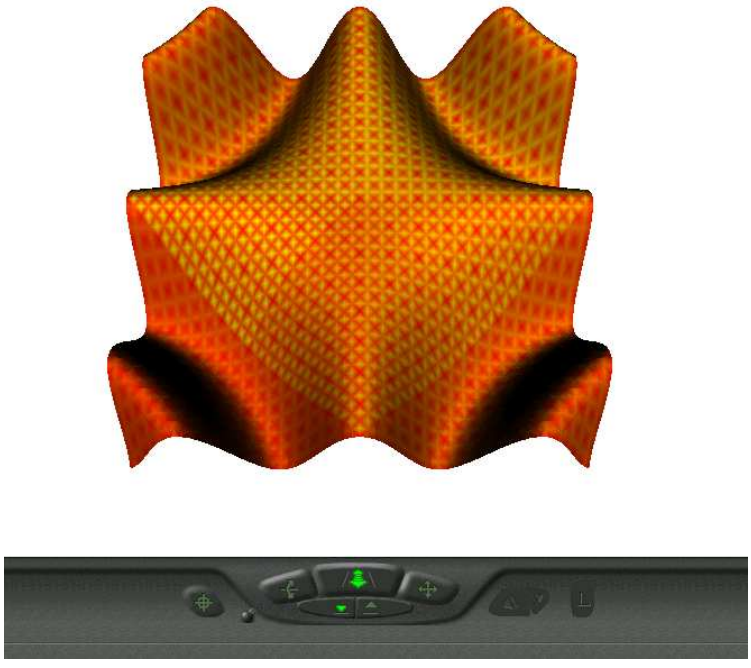


Figura 5.26: Puntos de vistas y niveles de detalle

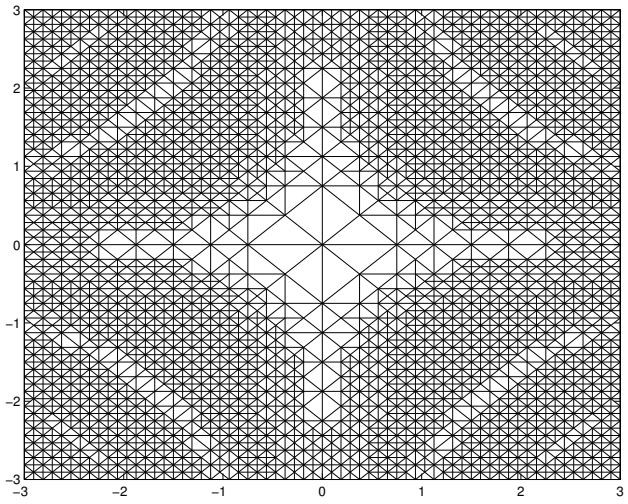


(a) Malla inicial 2D

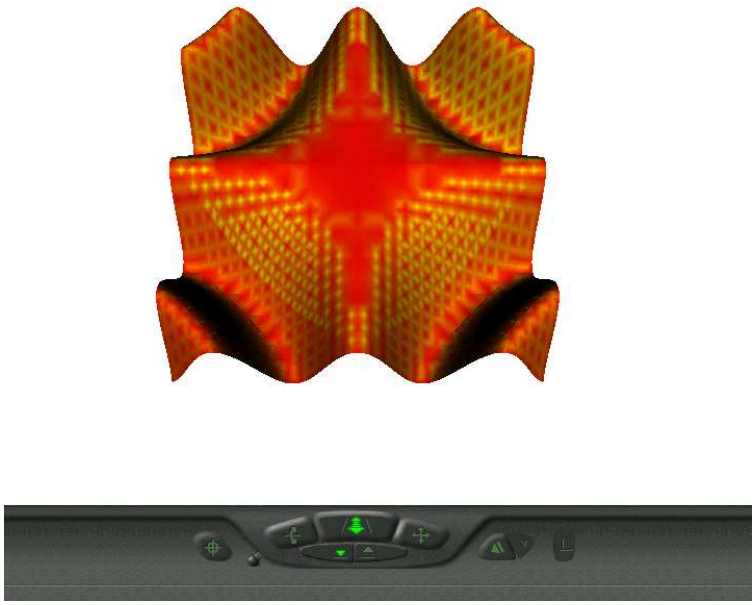


(b) VRML. LOD 1

Figura 5.27: Malla 2D y representación VRML con nivel de detalle 1

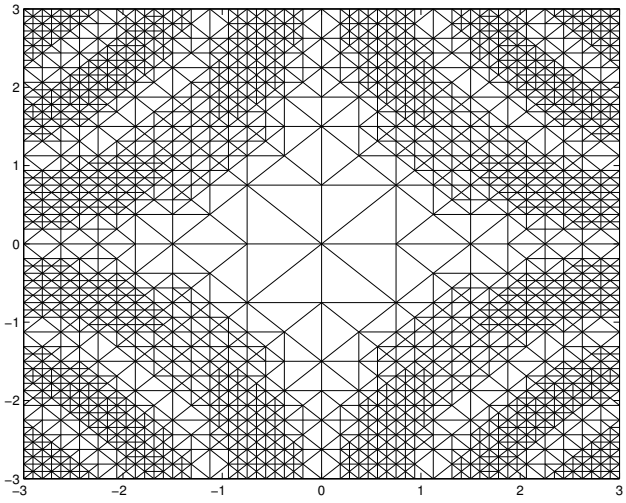


(a) Malla 2D

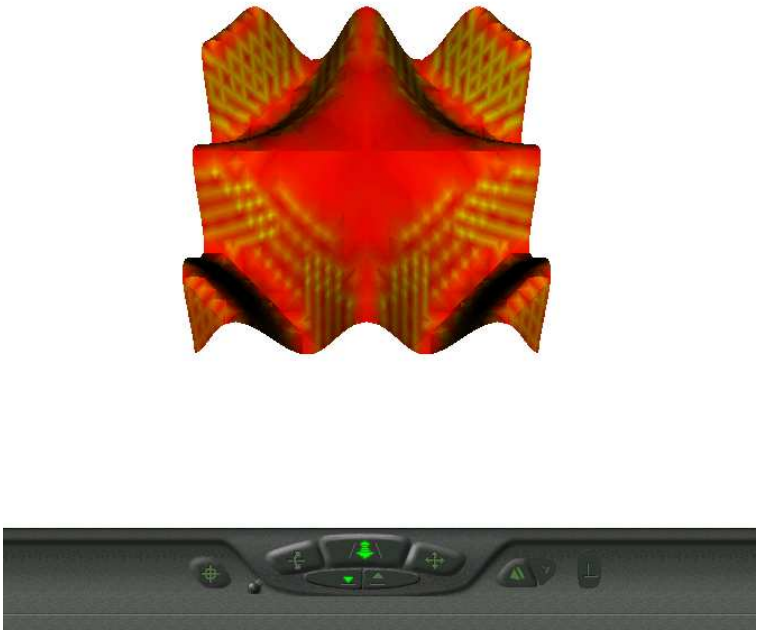


(b) VRML. LOD 2

Figura 5.28: Malla 2D y representación VRML con nivel de detalle 2 (10 % error)

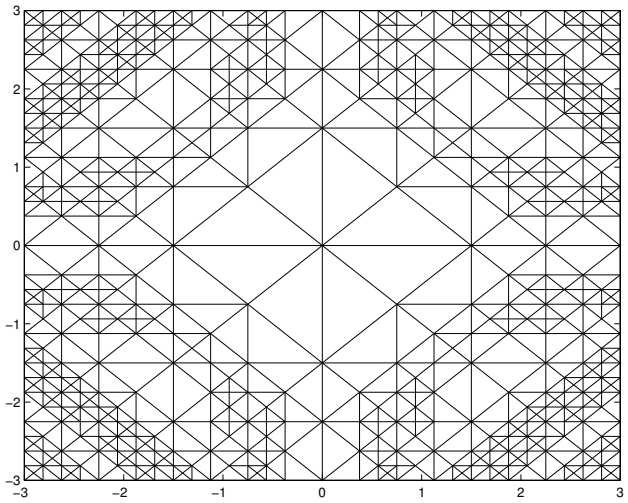


(a) Malla 2D

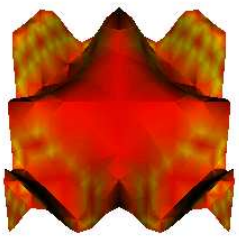


(b) VRML. LOD 3

Figura 5.29: Malla 2D y representación VRML con nivel de detalle 3 (30 % error)



(a) Malla 2D



(b) VRML LOD4

Figura 5.30: Malla 2D y representación VRML con nivel de detalle 4 (60 % error)

5.4. Método de Elementos Finitos

En esta sección se explica la incorporación del algoritmo de refinamiento 2D-SBR al código comercial de elementos finitos *PDE Toolbox* desarrollado en $Matlab^{TM}$ [52], y se aplica a la solución númerica de una ecuación en derivadas parciales en 2D.

El código de elementos finitos PDE Toolbox proporciona una serie de rutinas desarrolladas en Matlab que implementan el Método de Elementos Finitos y que el usuario puede ejecutar en la línea de comandos de Matlab. Para una referencia rápida del Método de Elementos Finitos ver la sección 1.2 de la página 1.2. El código de dichas rutinas es accesible al usuario pudiendo modificar y adaptarlo según las necesidades. Por otro lado, el PDE Toolbox proporciona una interfaz gráfica sencilla mediante un entorno de ventanas que permite configurar y resolver un problema de elementos finitos. Esta última característica es especialmente util para el usuario ya que permite dar entrada a la geometría de la malla, modificarla, etc. de una forma gráfica e interactiva. El PDE Toolbox y Matlab poseen unas características avanzadas de visualización en 2D y 3D que se adaptan muy bien a la simulación numérica.

El PDE Toolbox puede resolver diverentes variantes de ecuaciones y sistemas de ecuaciones: elíptica, parabólica, hiperbólica y ecuaciones con valores propios. Además considera modos de aplicación según el tipo de ecuación, entre otros se destacan, mecánica de estructuras, electrostática, magnetostática, trasferencia de calor y corriente alterna y continua. En cuanto a problemas no lineales, permite resolver la ecuación elípica no lineal utilizando el método de Gauss-Newton.

En la figura 5.31 se presenta un diagrama en bloques que resume la estructura en módulos de PDE Toolbox. Las formas elípticas corresponden a datos que se generan o se introducen al sistema, mientras que las formas cuadradas corresponden a procedimientos (conjunto de rutinas Matlab) que realizan alguna función en el flujo de resolución de una ecuación o sistema.

La incorporación del algoritmo 2D-SBR se realiza en el módulo de refinamiento de la malla. Concretamente la rutina $refinemesh.m$ en el PDE

Toolbox es la que realiza el proceso de refinamiento. Cabe destacar que la estructura de datos que almacena los datos de la malla en el PDE Toolbox está basada en nodos y triángulos, igual que el algoritmo 2D-SBR. Por otro lado, la entrada y la salida del algoritmo 2D-SBR se ha tenido que homogeneizar a la rutina del sistema $refinemesh.m$ para que la sustitución de la rutina sea trasparente al usuario y al sistema.

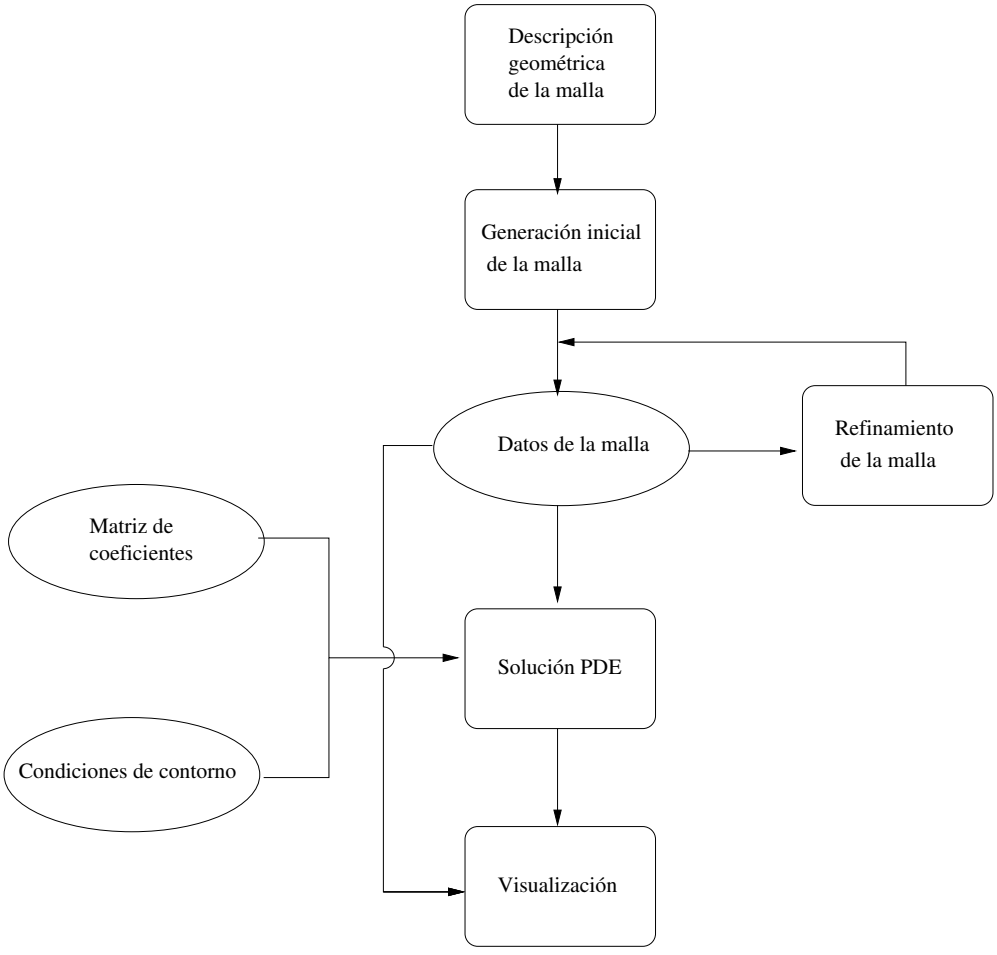


Figura 5.31: Diagrama en bloques de PDE Toolbox

La ecuación elíptica que resuelve el paquete es la siguiente:

$$-\nabla \cdot (c\nabla u) + au = f \quad en\ \Omega \tag{5.6}$$

La versión no lineal de la ecuación elíptica es la siguiente:

$$-\nabla \cdot (c(u)\nabla u) + a(u)u = f(u) \tag{5.7}$$

Además se presenta la forma básica de los otros tipos de ecuaciones que también permite resolver:

1. Ecuación parabólica:

$$d\frac{\partial u}{\partial t} - \nabla \cdot (c\nabla u) + au = f \tag{5.8}$$

2. Ecuación hiperbólica:

$$d\frac{\partial^2 u}{\partial^2 t} - \nabla \cdot (c\nabla u) + au = f \tag{5.9}$$

3. Problema de autovalores:

$$-\nabla \cdot (c\nabla u) + au = \lambda du \tag{5.10}$$

La idea básica para la resolución de ecuaciones no lineales es utilizar iteraciones del método Gauss-Newton. Tras plantear la formulación debil de un problema no lineal elíptico en la ecuación (5.7) y aplicar la fórmula de Green se calcula el vector residuo, resultando: $\rho(U) = (K + M + Q)U - (F + G)$, siendo las matrices *K, M, Q* y los vectores *F, G* resultantes de ensamblar el problema (5.7). Entendemos ensamblar el sistema por el cálculo de la contribución de las integrales aparecidas en la formulación debil a cada elemento de la malla, lo que se refleja en las matrices anteriores.

Sea U^n la aproximación a la solución del problema elíptico no lineal. Si U^n es una aproximación suficientemente buena a la solución U, entonces se calcula una siguiente aproximación U^{n+1} que se obtiene al resolver el sistema linealizado siguiente:

$$\left(\frac{\nabla \rho(U^n)}{\nabla U}\right)(U^{n+1} - U^n) = -\alpha\rho(U^n) \tag{5.11}$$

donde α es un número positivo.

Si α es suficientemente pequeño, entonces se cumple que:

$$||\rho(U^{n+1})|| \ < \ ||\rho(U^n)|| \tag{5.12}$$

y

$$p_n = \left(\frac{\nabla\rho(U^n)}{\nabla U}\right)^{-1} \rho(U^n) \tag{5.13}$$

se denomina dirección descendente para $||\rho(U)||$, donde $||\cdot||$ representa la norma l_2. Entonces, la iteración según el método de Gauss-Newton es:

$$U^{n+1} = U^n + \alpha\rho_n \tag{5.14}$$

y $\alpha \leq 1$ pero lo suficientemente cerca a 1 para que en cada paso de la iteración se tenga un descenso razonable que haga converger rápidamente a la solución.

Con respecto de la convergencia de este método hay que decir que el comportamiento es local y que se asegura convergencia cuando U^0 está cerca de la solución. Como esta restricción para la solución inicial puede ser demasiado fuerte, se considera otro tipo de valor para α que mejora la convergencia cuando la aproximación inicial no es buena. Se trata de la estrategia de búsqueda de Armijo-Goldstein que consiste en elegir el mayor coeficiente α de entre la secuencia 1, $\frac{1}{2}$, $\frac{1}{4}$, ..., tal que se cumpla la siguiente desigualdad:

$$||\rho(U^n)|| - ||\rho(U^n) + \alpha p_n|| \ \geq \ \frac{\alpha}{2}||\rho(U^n)|| \tag{5.15}$$

Para el cálculo del Jacobiano implicado en las fórmulas anteriores, el PDE Toolbox da la opción de aproximarlo según las siguientes posibilidades:

1. Diferencias finitas, lo cual da una opción no muy costosa y razonable para la mayoría de los problemas.

2. Usar como Jacobiano la matriz de rigidez K, quedando $U^{n+1} = K^{-1}F$. Esta solución aunque puede dar una convergencia lenta, es rápida y sencilla en el cálculo de las iteraciones.

3. Aproximar el Jacobiano como compromiso de las dos anteriores, resultando $J = K^c + M^{a-f} + diag + ((K^c + M^a)U)$, siendo K y M las matrices de rigidez y de masas respectivamente, y sus índices son los coeficientes en el ensamblaje del sistema.

Respecto al indicador de error $E(K)$ para un elemento K, éste se obtiene sumando la contribución de cada una de las aristas al elemento K [52]. Para la ecuación elíptica en (5.6), el estimador de error es:

$$E(K) = \alpha \|h(f - au)\|_K + \beta \Big(\frac{1}{2} \sum_{e \in \partial K} h_e^2 [\mathbf{n}_e \cdot c \nabla u_h]^2 \Big)^{\frac{1}{2}} \tag{5.16}$$

donde $\mathbf{n}_e$ es el vector normal unidad de la arista e y los coeficientes α y β son independientes de la triangulación usada.

5.4.1. Solución de un problema elíptico no lineal

Afrontamos en esta sección la solución de una ecuación diferencial elíptica no lineal, problema de Dirichlet, con el paquete PDE ToolbBox. La ecuación, tomada de [57], es la siguiente:

$$-\Delta u + u^3 = h \quad en\ \Omega \tag{5.17}$$

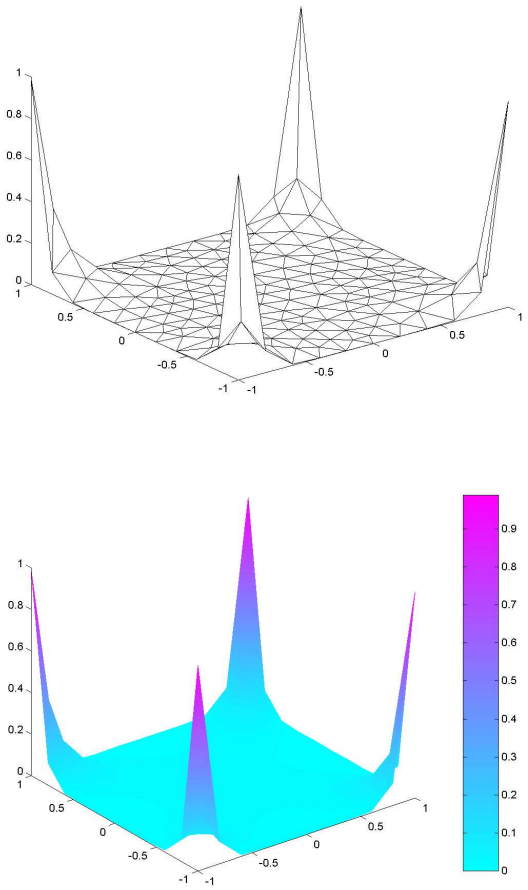
para Ω el recinto $[1, -1] \times [1, -1]$ y $h = -10(x \cdot y)^8 (x^2 + y^2) + (x \cdot y)^{30}$.

La solución exacta de (5.17) es:

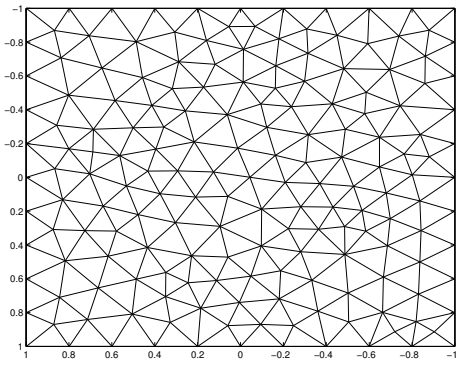
$$u = u_{ex} = (x \cdot y)^{10} \tag{5.18}$$

En la figura 5.32, presentamos la malla inicial de tipo Delaunay para resolver el problema y la solución del problema en esta malla inicial. Después de seis iteraciones se alcanza la solución con la precisión deseada de 5×10^{-7} y con un total de 55403 puntos y 109641 triángulos en la malla final.

Las figuras 5.33, 5.34, 5.35, 5.36 corresponden a los refinamientos locales uno, dos, tres y cuatro de la malla inicial usando el algoritmo 2D-SBR y la partición 4T-LE, y se presentan además las soluciones obtenidas respectivamente con el PDE Toolbox.

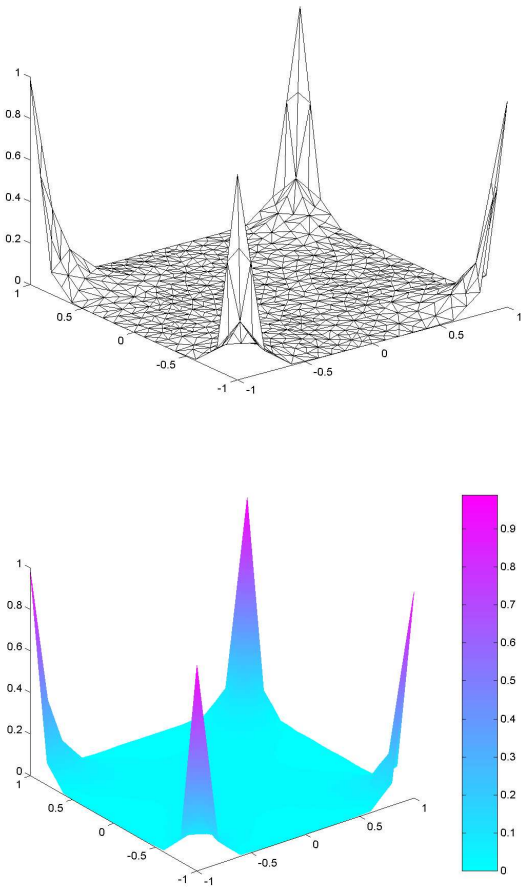


(a) Solución inicial

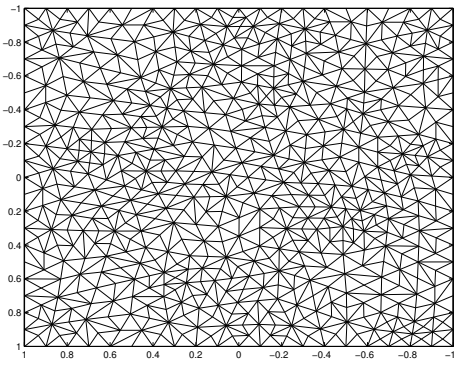


(b) Malla 2D inicial, 175 nodos, 308 triángulos

Figura 5.32: Solución inicial de la ecuación elíptica no lineal

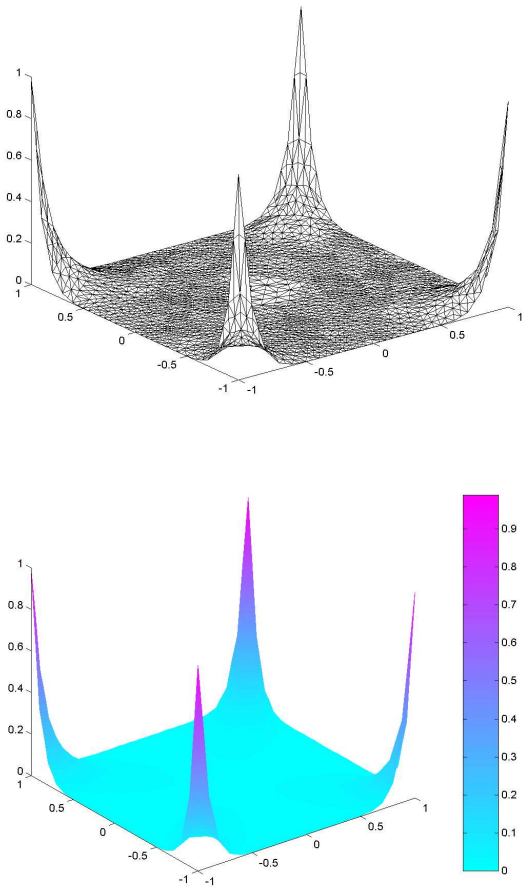


(a) Solución en el paso de refinamiento 1

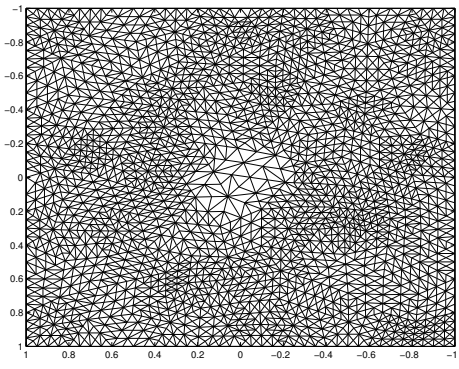


(b) Malla 2D, 657 nodos, 1232 triángulos

Figura 5.33: Solución en el nivel de refinamiento 1

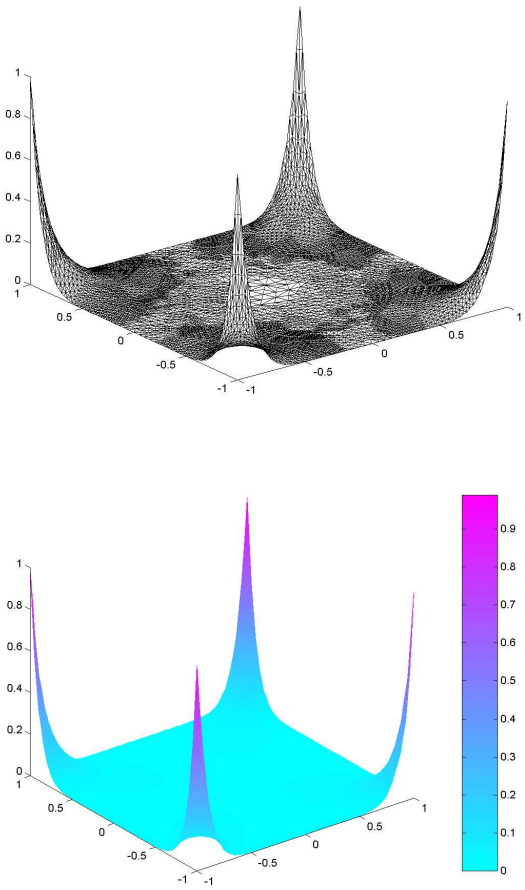


(a) Solución en el paso de refinamiento 2

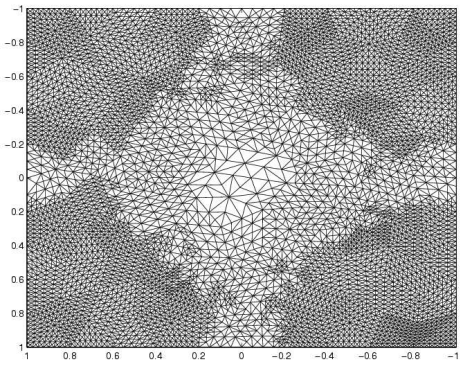


(b) Malla 2D, 2474 nodos, 4787 triángulos

Figura 5.34: Solución en el nivel de refinamiento 2

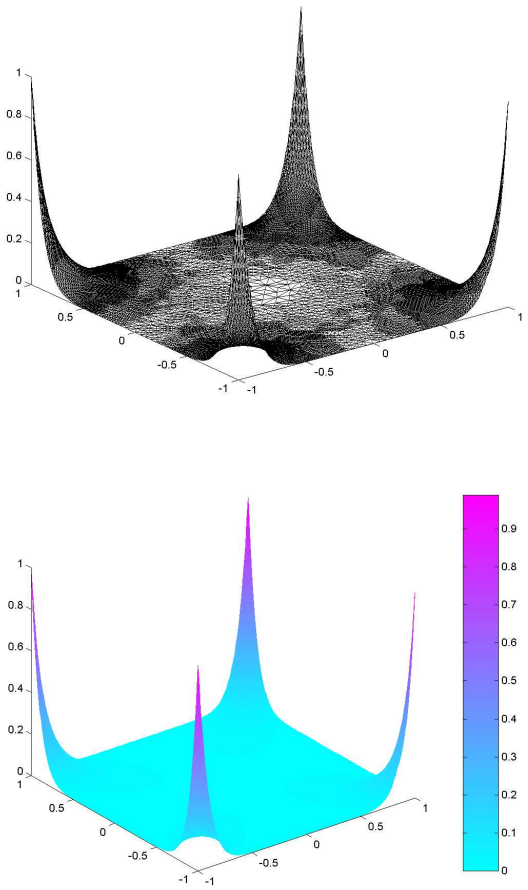


(a) Solución en el paso de refinamiento 3

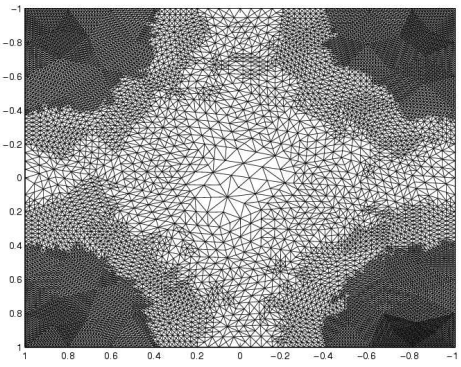


(b) Malla 2D, 6761 nodos, 13230 triángulos

Figura 5.35: Solución en el nivel de refinamiento 3



(a) Solución en el paso de refinamiento 4



(b) Malla 2D, 14782 nodos, 29075 triángulos

Figura 5.36: Solución en el nivel de refinamiento 4

Capítulo 6

Conclusiones y Trabajo Futuro

6.1. Conclusiones

El trabajo de investigación desarrollado en esta tesis permite establecer las siguientes conclusiones finales:

1. La estructura de datos tipo grafo permite un enfoque más natural de los algoritmos de refinamiento y desrefinamiento basado en el esqueleto en dimensión 2 y 3. Concretamente en dimensión 2 se tiene que:

   - El almacenamiento de la malla requiere de una lista de nodos y de una lista de triángulos en función de los nodos. Además, la estructura de datos tipo grafo permite realizar las operaciones de subdivisión de la malla de forma dinámica y con un coste de almacenamiento pequeño para el refinamiento o desrefinamiento local.
   - Se reducen las líneas de código. La nueva versión implementada en Matlab de los algoritmos en dimensión 2, reduce considerablemente el número de líneas de código de versiones anteriores. Además las nuevas versiones de los algoritmos ganan en simplicidad, claridad y flexibilidad.
   - La complejidad lineal (en el número de nodos) de los algoritmos en términos de tiempo permanece. Sin embargo, la complejidad en cuanto al almacenamiento se reduce drásticamente, ya que no se almacena la lista de aristas de la malla y se evita así la alta redundancia de información en estructuras de datos anteriores. Por

su parte, la estructura de datos tipo grafo es dinámica en el proceso de partición de la malla, no requieriendo costes excesivos, ni de almacenamiento ni de tiempo de cómputo.

2. El nuevo enfoque de los algoritmos 2D-SBR y 2D-SBD ha permitido estudiar y demostrar algunas propiedades geométricas de la partición 4T-LE cuando se aplican al refinamiento global de mallas no estructuradas.

   - El número de triángulos distintos generados es finito.
   - Se ha demostrado la mejora en cuanto al ángulo mínimo que manifiesta la partición 4T-LE.
   - Se ha estudiado el comportamiento asintótico de la partición 4T-LE en términos del grado de equilibrio: $\lim_{n\to\infty} gr(\tau_n) = 1$.

3. Se han aplicado los algoritmos a diversas áreas mostrando su versatilidad de uso y sus buenas propiedades. Concretamente:

   - Modelos Digitales del Terreno. El algoritmo de desrefinamiento se aplica a la aproximación de terrenos. Se propone así un nuevo tipo de aplicaciones de esta clase de algoritmos y de la partición 4T-LE.
   - Niveles de Detalle en Visualización. Aplicación del algoritmo de desrefinamiento en la generación de niveles de detalle en la visualización de superficies. Se ha incorporado la salida del algoritmo 2D-SBR en el código VRML proponiendo una solución alternativa y viable al modelo propio de VRML de generación de niveles de detalle.
   - Se han incorporado los algoritmos a un código de elementos finitos comercial y se ha resuelto a modo de ejemplo un problema elípitico no lineal en 2D.

6.2. Trabajo futuro

Asimismo, el periodo de investigación ha revelado nuevas y futuras áreas de actuación. Resumimos a continuación algunas de ellas:

1. Implementación en el caso de dimensión 3 del algortimo 3D-SBR con la estructura de datos basada en el grafo G^2.

2. Un problema crucial de los algoritmos que tratan con mallas, bien para refinar, desrefinar, visualizar etc. es que cuando el conjunto de datos es del orden de millones de elementos, el tiempo de ejecución degenera el rendimiento para aplicaciones de tiempo real. Por ejemplo, estrategias de partición de dominios en dimensión 3 y arquitecturas paralelas son un campo de actuación importante.

3. Los algoritmos de refinamiento y desrefinamiento pueden ser aplicados por ejemplo, a la compresión de mallas, mallas progresivas, reconocimiento de formas, tratamiento de imágenes, aproximación de superficies, etc., como ya tratan otros trabajos, por ejemplo [17, 39]. Los algoritmos de refinamiento y desrefinamiento son herramientas muy versátiles en áreas como simulación, gráficos por ordenador, geometría computacional, etc.

Apéndice A

Publicaciones

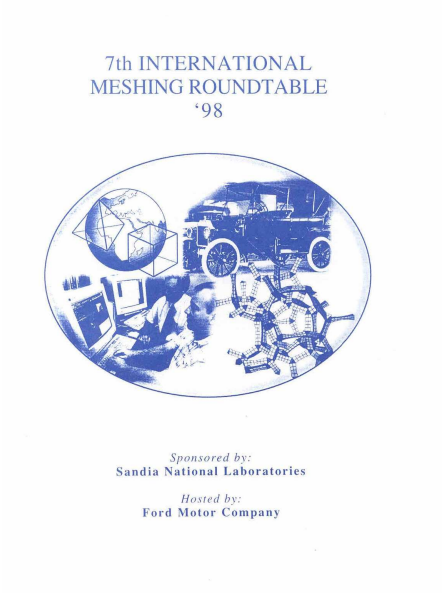
Se adjuntan en este apéndice las tres publicaciones más importantes a que ha dado lugar hasta el momento esta tesis:

1. A. Plaza, J.P. Suárez and M.A. Padrón, *Mesh graph structure for longest-edge refinement algorithms*, in Proceedings 7th International Meshing Roundtable'98, October, 1998 Michigan, SANDIA Report SAND 98-2250, pp. 335-344. [70].

2. J.P. Suárez, G.F. Carey, A. Plaza, and M.A. Padrón, *Graph based data structures for skeleton based refinement algorithms*, TICAM Technical Report, 01-10, Abril, 2001. [94].

3. J.P. Suárez, G.F. Carey and A. Plaza, *Graph based data structures for skeleton based refinement algorithms*, Comm. Num. Meth. Eng., Aceptado, Mayo, (2001). [95].

A.1. 7th International Meshing Roundtable'98



Mesh graph structure for longest-edge refinement algorithms

Angel Plaza, José P. Suárez, Miguel A. Padrón
Department of Mathematics. University of Las Palmas de Gran Canaria. Spain
{angel, jsuarez, mpadron}@aries.dma.ulpgc.es

Abstract. In this paper we introduce and discuss a graph structure associated with longest-edge algorithms (algorithms based on the bisection of triangles by the longest-edge) which can be used to reformulate longest-edge algorithms to develop new algorithms and to design efficient data structures for the refinement/derefinement of 2D and 3D triangulations.

Keywords. Longest-edge refinement, bisection, data structures, improvement.

1. Introduction

Modern finite element applications make extensive use of adaptive techniques to optimize the number of unknowns with respect to the accuracy of the numerical solutions. For this purpose, the underlying discretization mesh must be locally refined in regions where improved accuracy is needed. Moreover, multigrid or multilevel methods have shown to be of optimal or nearly optimal complexity for the solution of discrete systems arising from a wide range of partial differential equations. Since these methods are based on discretization hierarchies obtained from succesively refined meshes, they are very appropiate to be embedded in the adaptive framework.

The management of adaptively refined grids and the implementation of adaptive solvers require of specialized refinement and/or derefinement techniques. In particular, the local refinement and/or derefinement of the mesh should not involve a complete reconstruction of the data structures but only the local reconstruction of the grid is implied in the process. In adition, the local reconstruction work should remain proportional to the number of modified elements. To reach these goals, data structures which fit the structure of the problem as well as the refinement scheme are needed.

In the adaptive refinement/derefinement context longest-edge algorithms for triangulations have been extensively studied in the last 15 years (Rivara [12-17], Ferragut [4], Plaza *et al.* [11]). In particular Rivara has studied two algorithms based on longest-edge bisection of triangles have been discussed: the pure longest-edge algorithm (which only performs longest-edge bisection of the current triangles); and the 4-Triangles algorithm where for each target triangle, a longest-edge bisection is firstly performed and the newly vertex is used to subdivide the initial triangles in four. Rivara and Levin have generalized the pure longest-edge agorithm to 3 dimensions [12], while Plaza and Carey [10] developed a new refinement algorithm for tetrahedral grids based on the skeleton: the set comprissed by the faces of the tetrahedra. Their algorithm can be extended to a general dimension N. This algorithm has the advantaje of establishing a finite number of son-elements in the refinement process, and can be seen as the generalization to three dimensions of the 4-T two-dimensional refinement algorithm of Rivara. Other refinement algorithms based on bisection, developed in the last years for three dimensions can be found in [8],[9],[10].

The great amount of information to be managed in 3D case and the obvious complexity to deal with these meshes makes necessary to develop efficient and good algorithms and related data structures in terms of low storage and low computational cost. This fact has encouraged us to first, deal with the two-dimesional case to find out good data structures that improve the existing algorithms and secondly, apply these concepts and results to the three-dimensional case and maybe to a higher dimension as Plaza and Carey stated in [10].

The idea in [10] from Plaza and Carey to represent a mesh as an oriented graph have been taken in this paper as a good way to make easy the process of longest side bisection of Rivara. Besides, this idea considerably reduces the amount of data to be stored, as it will be shown in this paper. On the other hand, a data structure based on trees is introduced. Trees make it possible to represent the nestedness of the mesh in a hierarchical way, which benefits the multigrid methods and the refinement or derefinement of the mesh. Leinen in [6] made use of trees in a similar way for the two dimensional case and considered the posibility of extending to the three-dimensional case. Bey in [3] also relied on this structure and presented an algorithm for grid refinement which was implemented in the AGM^{3D} software.

In this paper, we present some data structures based on graphs and trees that fit the longest side bisection algorithms of Rivara. In addition, we consider some properties that let us prove the goodness of these data structures and the finiteness for that class of algorithms.

2. Longest edge refinement algorithms.

Definitions:
The *longest edge bisection* of a triangle **t** is the partition of the triangle by the midpoint of its longest edge and the opposite vertex. The *neighbor* of **t** is the neighboring triangle $\mathbf{t}^*$ which shares with **t** the longest edge of **t**. We say that a mesh is *conforming* if any adjacent element (triangle in 2D or tetrahedron in 3D) share an entire face (3D) or an edge (2D) or a common vertex.

2.1 Forward Longest Edge Bisection

In order to make a grid conforming, the local refinement of a given triangle involves refinement of the triangle itself and refinement of some of its neighbors. The algorithms bisect a triangle t (thin line in Fig.1) and its neighbors or the boundary of the domain is reached and so on iteratively until the last two triangles share the same longest edge. Although may be obvius that the refinement propagation stops, we prove later the finiteness of the process. The same idea has to be applied in order to conform the set of non-conforming points generated, (dashed line in Fig. 1) in the inverse of the order in which they were created.

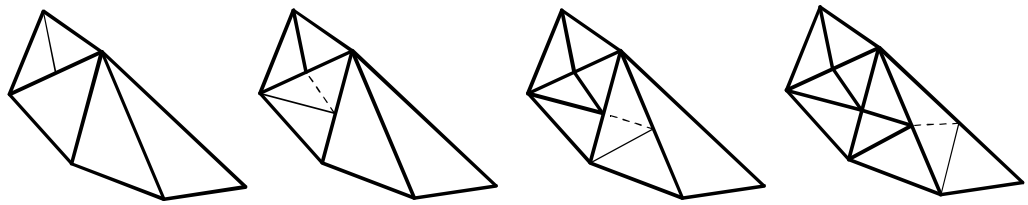


Fig. 1 .Forward Longest Edge Bisection

We shall call this version of the algorithm the Forward Longest Edge Bisection (FLEB).

2.2 Backward Longest Edge Bisection

In [16] Rivara has introduced an improved version of the algorithm called Backward Longest Edge Bisection (BLEB).

Longest-Edge Propagation Path (LEPP) defined by Rivara in [17] is an ordered list of all the triangles $\{\mathbf{t_0},\mathbf{t_1},\ldots,\mathbf{t_n}\}$ such that $\mathbf{t_i}$ is the neighboor triangle of $\mathbf{t_{i-1}}$ by the longest edge of $\mathbf{t_{i-1}}$.

Given a triangle, that last version runs the **LEPP** and it only bisects one or two of the last triangles of the path. The process is repeated as many times as needed until the original triangle is also bisected.

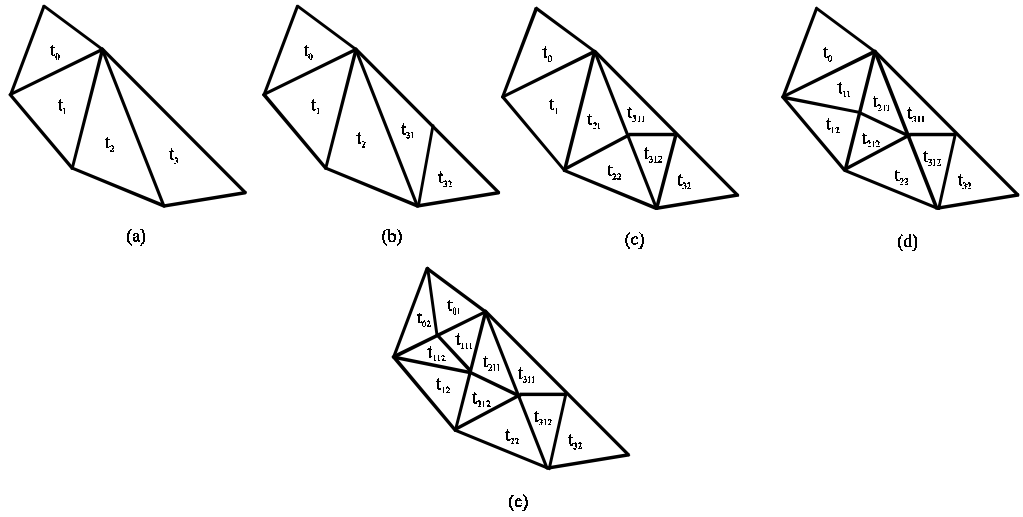


Fig. 2. Backward Longest-Edge Bisection of triangle t_0

In Fig. 2 the Backward Longest-Edge Bisection has been applied to the original triangulation (Fig. 2.a). In the figure are represented the intermediate meshes while bisection is applied. As an explanation see also the **LEPP** of $\mathbf{t_0}$ in each step of the algorithm in the following table.

LEPP(t_0)	Triangles to be bisected	Figure
$\{t_0,t_1,t_2,t_3\}$	$\{t_3\}$	2.(a)
$\{t_0,t_1,t_2,t_{31}\}$	$\{t_2,t_{31}\}$	2.(b)
$\{t_0,t_1,t_{21}\}$	$\{t_1,t_{21}\}$	2.(c)
$\{t_0,t_{11}\}$	$\{t_0,t_{11}\}$	2.(d)

At this point, the following remarks are in order:

Remarks:

1. Both algorithms are equivalent in the sense that they produce the same refined mesh in the end.
2. In both algorithms, the concept of the Longest-Edge Propagation Path (LEPP) introduced above has been repeatedly used. In the first one it is used implicitly and in the second one explicitly.
3. In the first algorithm, FLEB, the assurance of conformity is carried out in each step, however, in the second one, BLEB, nothing must be done because this version guarantees conformity.
4. For both algorithms, a suitable data structure that explicity manages the neighbor-triangle relation should be used.
5. It is expected that both algorithms stop whatever mesh you use. For this purpose, it is sufficient to prove that the LEPP is finite in each step and has no loops.

3. Triangle edges classification and the oriented graph.

The algorithms based on the longest-edge of a triangle discussed above have the advantage that the different possibilities for refining a triangle depend only on determining the longest edge. In general, we can say that these algorithms classify the edges of each triangle in two types, the longest one, **type 1**, and the remaining two edges, **type 2**. Therefore, it is supposed that this classification can be done so that the algorithms perform the right job.

Based on this idea, Plaza and Carey in [10] proposed to represent this classification as an oriented graph in which a vertex of the graph represents an edge of the triangle and an edge of the graph between nodes **a** and **b** represents the relation "*length of edge x in the triangle is less than length of edge y*". See Fig. 3. In [7], this classification was used to explain the number of different possible configurations obtained by the 3-dimensional refinement algorithm proposed in that paper.

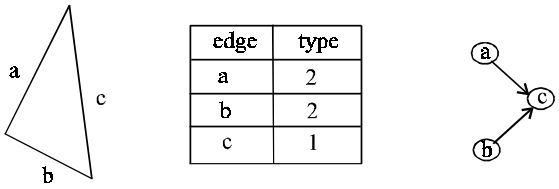


Fig. 3. Triangle, edges types and associated graph

This idea can be beneralized as follows to the whole mesh.

Definition:

Let τ *be any mesh having a finite number of triangles. For each triangle* t *in* τ*, we sort the edges of* t *as above and then build the oriented graph* $G(V,E)$ *associated to the whole mesh, where:*

- $V = \{v_0, v_1, \ldots, v_n\}$ *so that* $v_i \in V$ *is the vertex on the graph representing an edge of the triangle in the mesh.*
- $E = \{e_0, e_1, \ldots e_m\}$ *so that* $e_j \in E$ *is the edge on the graph representing the relation* R *between two vertices belonging to* v *so that* R *is "length less than". That is, e* R *f if and only if:*
  - *i) e and f belong to the same triangle*
  - *ii) length(e) < length(f)*

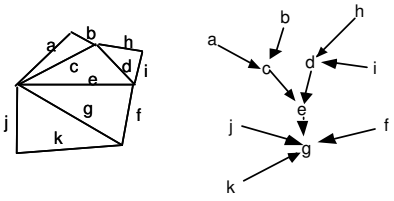


Fig. 4

Note hovewer that, when regular elements appear in the mesh, and since for such elements the longest-edge is not unique, the associated graph can include undesirable loops. A critical situation is illustrated in Fig. 5 where all the triangles are regular and as a consequence the classification of the edges in each triangle can not b e carried out.

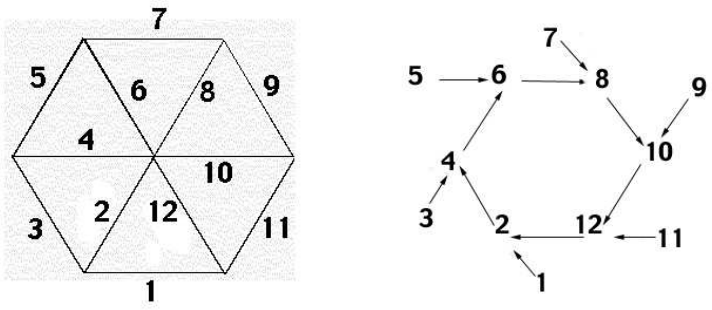


Fig. 5

In order to avoid the ambigousness caused by non-unique longest-edges in the pratical implementation of the longes-edge algorithms the following additional convection is used: whenever the *LEPP(t)* is not unique (due to the existence of elements having non-unique longest-edges) the shortest possible path is selected.

Remark. *Let* Ω *be any boundary 2-dimensional domain, with poligonal boundary* Γ*, and let* τ *be any unstructured and conforming triangulation of* Ω *having regular triangles. Then, the following conditions have to be taken into account in building the associated graph of* τ*:*

1. *If a regular triangle* $t \in \tau$ *is the first to be considered in a LEPP of the graph, we convey any possible relation among the edges.*
2. *Let* t_i *and* t_{i+1} *be two neighbor triangles* $\in \tau$ *to be considered in a LEPP of the graph, if* t_{i+1} *is regular, we make the edges of* t_{i+1} *to point to the shared edge in the associated graph.*

Fig. 6

Proposition 1.- *The previous remark allows us to deal with regular triangles without ambiguity.*
Proposition 2.- *Let* $G(V,E)$ *be the associated graph of an unstructured and conforming triangulation* τ*, then* G *does not hold loops.*
Proof:
There are two situations in which loops can occur. The first one arises in meshes like those referred in Prop. 1, Fig. 5. In this case, the way in which the graph is built assures the non-existence of loops (see Prop. 1). The second situation is as follows:

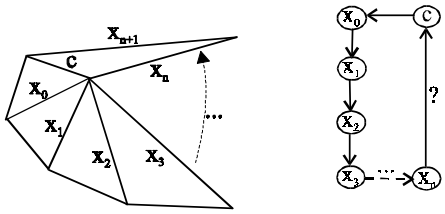


Fig. 7. Triangulation and simplified associated graph in which a possible loop can occur.

In which:

- **I.** $\mathbf{X=\{x_0, x_1, x_2, x_3, x_n\}}$ are the common edges shared by the neighbor triangles $\in$ **LEPP** of $\mathbf{t_0}$ ($\mathbf{t_0}$ is the triangle defined by edges **c** and x_0)
- **II.** $\mathbf{Length(x_{i-1}) < Length(x_i)}$, for i=1 to n

In that situation we must prove that $\mathbf{x_n}$ does not point to **c**, otherwise there is a loop. It can be obvious in Fig.7 that **Length(c)** is less than both $\mathbf{Length(x_n)}$ and $\mathbf{Length(x_{n+1})}$ and so it is impossible that $\mathbf{x_n}$ or $\mathbf{x_{n+1}}$ points to **c**.

Anyway, let us suppose that there is a loop from $\mathbf{x_n}$ to **c** and then we will yield an absurd caused by the wrong supposition.

If so,

- **III.** $\mathbf{Length(c) > Length(x_n)}$

But we know that

- **IV.** $\mathbf{Length(x_0) > Length(c)}$

Then, from III and IV we have:

- **V.** $\mathbf{Length(x_0) > Length(x_n)}$

However, V is not possible because we supposed II. This is due to the supposition in III. Consequently, it must be:

Length(c) < Length(x_n)

and then there are not loops because:

- **c** and x_{n+1} point to x_n (if x_n is of **type 1**) Fig. 8.a or
- **c** and x_n point to x_{n+1} (if x_{n+1} is of **type 1**) Fig. 8.b

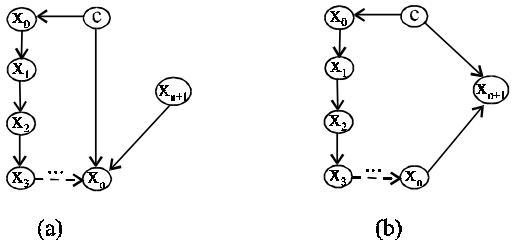


Fig. 8

4. A graph data structure

In what follows a data structure based on the graph representation of the mesh is proposed.

Remark. *In essence, the goal is to design a data structure that provides the best balance between storage and computational cost without increasing significantly the code complexity.*

Although that data structure could make the bisection process easy and have good features related with the implementation, we also need another data structure that let us preserve mesh nestedness, in other words, mesh genealogy. This last requirement benefits refinement or derefinement processes as well as multigrid methods, as we pointed out in the introduction. Moreover, we must to have the last remark in mind. Several solutions have been proposed for that purpose but trees seem suit best, as many authors think: Carey *et al.* in [4], Leinen in [6] or Bey in [3].

4.1 Modified Adjacency List

A standard data structure to model an oriented graph is the *Adjacency List*, especially adequate to the case when few links among the vertices are managed [1], [5]. In such a list, the entire set of vertices in the graph are arranged as a linked list (see column linked list in Fig. 9.c). It can be noted that instead of using a linked list for all edges in the mesh, an array can also be used. This benefits the direct access to a given item, which is not achieved by means of a linked list. However, an additional and fixed cost of storage is incurred on choosing the array structure. Furthermore, the advantage of a dynamic structure based on links and pointers is lost. In this paper we choose the first structure (linked list) but it is an open decision, which depends on the particular implementation.

For each node in the list, a new linked list is associated (see row linked lists in Fig. 9.c). It represents the vertices adjacent to the first one. That last adjaceny list is based on triangle edges, and so, we call it *Edge Adjacency List.*

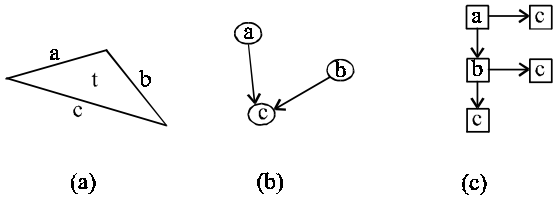


Fig. 9. (a) An arbitrary triangle. (b) Graph associated. (c) Adjacency List

Up to now our data structure as well as the graph are based on edges. However, it is desirable to deal with triangles as implicitly Rivara's algorithms do. Therefore, we introduce a new list but based on triangles. The column linked list holding the vertices in the graph is preserved. For each vertex in the graph having two incident edges, we link a new node, in a row linked list, representing the triangle defined by them (see Fig. 10). We call this new list *Triangle Adjacency List*.

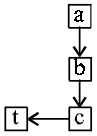


Fig 10. The new list based on triangles. See Fig. 9

It really represents a triangle related with its longest edge, except for the case described in Remarx 2. By means of that new list for each vertex, we are able to access not only the edges in the triangles but the triangles in the mesh in a quick and easy way.

Combining now the two lists in only one yield the *Modified Adjacency List*.

Definition:

Let τ *be an unstructured and conforming triangulation. Let* $G(V,E)$ *be the associated graph representing a classification based on the triangle longest edge except for the case of regular elements. We introduce the Modified Adjacency List, a data structure that make it possible to access both edges and triangles as Rivara's algorithms need.*

Graphically:

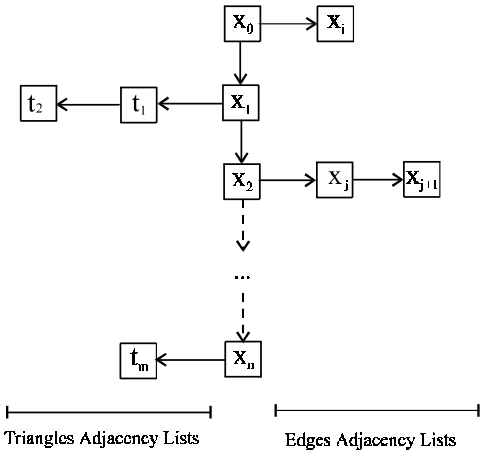


Fig. 11. Modified Adjacency List

Remark:

1. The maximum number of nodes in each of the *Edge Adjacency Lists* is 2.
2. The maximum number of nodes in each of the *Triangle Adjacency Lists is 2.*
3. Given an arbitrary edge, the maximum number of nodes in its *Edge Adjacency List* plus its *Triangle Adjacency List* is 2.

Proposition 3.-*Let* n *be the number of edges in a triangulation* τ*, then the storage cost needed to store the Modified Adjacency List is at most 3n.*

Proposition 4.- *The minimum maintenance cost of Modified Adjacency List holds BLSB and affects only to five edges (if two triangles are bisected) or three (if one triangle is bisected) in each refinement step.*

Remark: *The previous proposition ensures the locality of the Modified Adjacency List data structure.*

Proposition 5.- *Let* τ *be an unstructured and conforming triangulation* τ*, let* e_i *be one of the three edges of an arbitrary triangle* t_j *of* τ *and let* N *be the number of triangles in* τ*, then Rivara's* $LEPP(t_j)$ *can be obtained from the depth run of the Modified Adjacency List from* e_i*, with a maximum cost of (N+1).*

Proposition 6.- *Rivara's algorithms stop in a finite number of steps.*
Proof:
As mentioned in Remark 1.2, Rivara's algorithms deal with LEPP in order to make triangles bisections. Provided that **LEPP** was always finite, we will ensure the finiteness of algorithms. In fact, **LEPP** is finite because **G(V,E)** does not hold loops (see Proposition 2). An upper bound of items in LEPP is **N**, where **N** is the number of triangles in the mesh.

4.2 Trees to store mesh genealogy.

As pointed out above in section 4, trees appear to be a suitable data structure for representing nestedness in the mesh. The typical order relation on trees is not assumed, but another one that stands for nestedness. Therefore, if a triangle t_1 is subdivided in two, denoted by t_{11} and t_{12}, we can describe that relation as:

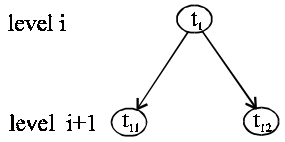


Fig. 12. A tree representing a triangle subdivision.

We now study how trees can be adapted for using them in Rivara's algorithms and generally in algorithms based on bisection. One desirable restriction to be satisfied by trees is that nodes of a given level have to represent conforming triangles and only just the subtriangles created in a given level of the refinement.

Look at Fig. 1 related to FLEB and notice that in each level of refinement, one given triangle can be subdivided in three adittional triangles. However, if we consider BLEB in Fig. 2, two only subtriangles appear in each step of the refinement. This let us to think of one data structure based on a ternary tree for FLEB and a binary tree for BLEB. Binary trees are well known and have been studied by several authors [1].

We concentrate on Rivara BLEB's because it presents the simplest case. The division process of BLEB can be modelled as a binary tree like in Fig. 12.

Definitions:
A *node* on the tree represents one triangle in the mesh and a top down *arrow* represents a relationship father-son. A node is said a *father* from other one named *son* if the last is obtainded by the bisection of the father node. A node is named *root* if it is the first on the tree. A *level* on the tree represents just a refinement level, and it is enumerated like in Fig. 12.

Remark:

1. *Leaf* nodes on the tree represent non-bisected triangles available in the mesh and *nonleaf* nodes stand for subdivided triangles.
2. To model the entire mesh we have to consider as many tre es as triangles in the original mesh, each one having a given root standing for the triangles.
3. Storing the mesh in this hierarchical way makes it possible to check the refinement level. In this sense, a well balanced tree means a uniform refined-region.
4. Provided that triangle sizes in the mesh are similar, trees also offer a way to control mesh smoothness.
5. The neighbors of one triangle can be obtained accessing at most 3 different subtrees, which, as pointed out in Remark 5 for the Modified Adjacency List, also ensures the locality of that data structure.

5. Conclusions

We have reviewed longest-edge algorithms for local refinement of triangular computational meshes and we have proposed an associated graph structure to provide an efficient framework for implementation of these algorithms.

We have focused on Rivara's algorithms in 2D, Forward and Backward Longest Edge Bisection. Although a great number of jobs related to algorithms in 2D are known, the data structures involved, which really set up the kernel of algorithms, have received far less attention.

We think that whatever effort made in that direction will help building software related to this gruelling field of numerical treatment of partial differential equations, representation of regions or computer graphics. Moreover, we think that most of ideas shown here in 2D are aplicable to higher dimension.

The idea pointed out by Plaza and Carey in [10] related to a graph representing the edge classification in a triangle is taken here in order to design a new data structure which let us, in an efficient and easy way, run **LEPP** in both Forward and Backard Longest Edge Bisection of Rivara. Furthermore, we consider some properties that let us prove the goodness of these data structures and the finiteness of that class of algorithms.

Finally, we briefly describe another data strcuture based on trees which make it possible to store the hierarchical way in which the grid is refined or derefined.

Special thanks to Maria Cecilia Rivara for her helpful suggestions and kind support in the preparation of this paper.

6. References

[1] A. V. Aho, J.E. Hopcroft y J.D. Ullman, "*Data Structures and Algorithms*", Addison-Wesley, (1983).

[2] J. Bey, "*Tetrahedral grid refinement*", Computing **55**, 355-378 (1995).

[3] G.F. Carey, M. Sharma, K.C. Wang, A. Pardhanani, *"Some aspects of adaptive grid computations"*, Computer & Structures, Vol. **30**, Nº 1/2, 297-302, (1988).

[4] L. Ferragut, NEPTUNO, "*A system of adaptive finite element method"* , Dept. Mat. Aplic. y Met. Inf., ETSI Minas, Madrid, (1987).

[5] F. Harary, "*Graph Theory"*, Addison-Wesley, (1972).

[6] P. Leinen, *"Data structures and concepts for adaptive finite element methods"*, Computing **55**, 325-354 (1995).

[7] A. Liu and B. Joe, "*Quality local refinement of tetrahedral meshes based on bisection"*, SIAM J. Sci. Comput., **16**, 1269-1291, (1995).

[8] A. Mukherjee, *"An adaptive finite element code for elliptic boundary value problems in three dimensions with applications in numerical relativity"*, Phd. Thesis, Penn. State University, University Park, PA 16802, (1996).

[9] E. Muthukrishnan, P.S. Shiakolas, R.V. Nambiar and K.L. Lawrence, *"Simple algorithm for Adaptive Refinement of three-dimensional finite element tetrahedral meshes"* , AIAA Journal, **33**, 928-932, (1995).

[10] A. Plaza and G.F. Carey, *"A new refinement algorithm for tetrahedral grids based on skeleton"*, TICAM Report 96-55 , November, (1996).

[11] A. Plaza, R. Montenegro and L. Ferragut, *"An improved derefinement algorithm of nested meshes"*, in Advances in Post and Preprocesing for Finite Element Technology, M. Papadrakakis ed., Civil-Comp Ltd., 175-180, (1994).

[12] M.C. Rivara and C. Levin, "*A 3-D refinement algorithm suitable for adaptive and multi-grid techniques"*, Comm. in App. Num. Meth., **8**, 281-290, (1992).

[13] M.C. Rivara, "*Mesh refinement based on the generalized bisection of simplices"*, SIAM J. Numer. Anal., **2**, 604-613, (1984).

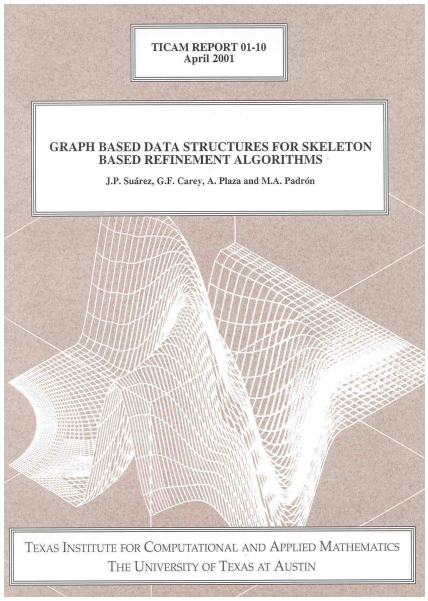
[14] M.C. Rivara, "*A grid generator based on 4-triangles conforming mesh refinement algorithms"*, Int. J. Num. Meth. Eng., **24**, 1343-1354, (1987).

[15] M.C. Rivara, "*Local modification of meshes for adaptive and/or multigrid finite-element methods"*, J. Comp. and Appl. Math., **36**, 79-89, (1991).

[16] M.C. Rivara, *"New mathematical tools and techniques for the refinement and/or improvement of unstructured triangulations"*, 5th. International Meshing Roundtable'96, Sandia National Laboratories, 77-86, (1996).

[17] M.C. Rivara, *"New longest-edge algorithms for the refinement and/or improvement of unstructured triangulations"*, Int. J. Num. Meth. Eng., **40**, 3313-3324, (1997).

A.2. TICAM Technical Report



GRAPH BASED DATA STRUCTURES FOR SKELETON BASED REFINEMENT ALGORITHMS

J. P. Suárez [1], G.F. Carey [2], A. Plaza [1] and M. A. Padrón [1]

[1] University of Las Palmas de Gran Canaria, Canary Islands, Spain, josepa@cicei.ulpgc.es,
[2] University of Texas at Austin, Austin, Texas, U.S.A., carey@cfdlab.ae.utexas.edu

ABSTRACT

In this paper we discuss a class of adaptive refinement algorithms for generating unstructured meshes in two and three dimensions. We focus on Skeleton Based Refinement (SBR) algorithms as proposed by Plaza and Carey [25] and provide an extension that involves the introduction of the graph of the skeleton for meshes consisting of simplex cells. By the use of data structures derived from the graph of the skeleton, we reformulate the Skeleton Based Refinement scheme and devise a more natural and consistent approach for this class of adaptive refinement algorithms. As an illustrative case, we discuss in detail the graphs for 2D refinement of triangulations and for 3D we propose a corresponding new face-based data structure for tetrahedra. Experiments using the two dimensional algorithm and exploring the properties of the associated graph are provided.

Keywords: Mesh refinement, skeleton, data structures, edge bisection.

1. INTRODUCTION

Refinement and coarsening algorithms and their supporting data structures are an essential part of adaptive mesh strategies for efficient reliable computations. Moreover, since the solution on a given mesh provides an appropriate starting iterate for the solution on the adapted mesh, these adaptive refinement and coarsening strategies have led to an increased interest in data structures to support iterative solution techniques and multigrid methods on nested grids. For example, see Carey [6] for a general treatment of computational grids and related topics such as mesh generation, adaptation and solution strategies. Refinement and coarsening algorithms can also be used for other purposes such as data mining for computer visualization (eg. see Carey *et al.* [7] [8]).

A critical feature of such algorithms, is the ability to manage the mesh effectively. The complexity of these meshes implies a need for efficient and robust algorithms having data structures with low storage overhead and low computational cost. Detailed features pertaining to the data structures have been described for specific parts of the adaptive finite element analysis process, such as searching during the mesh generation [5][18], refinement of existing meshes [9][13] and the solution process [18][19].

The data structure is also critical to other applications where complex meshes arise such as solid modeling. A review of edge data structures for solid modeling, that are relevant to the present work can be found in [34]. There are several libraries of algorithms and data structures in the published work from Computational Geometry that deal with this topic. These include, for example, the Computational Geometry Algorithms Library, CGAL [14] which uses object oriented and generic programming paradigms to implement combinatorial structures. Another library of similar type is the Library of Efficient Data Types and Algorithms, LEDA [14]. In this latter case, a general data structure for graphs and for planar maps can be derived from a graph data structure.

The skeleton view of a mesh provides a hierarchical set of simplex components defining the mesh in a recursive manner as the dimension is decreased. Thus, a mesh of tetrahedra can be defined using a skeleton made up of the surface triangles in the tessellation, these triangles can be viewed as composed of a skeleton of edges, which in turn are composed of a skeleton of nodes. In [25] Plaza and Carey first developed algorithms for grid refinement based on the skeleton in two and three dimensions, and refer to them respectively as two

Dimensional Skeleton Based Refinement (2D-SBR) and three Dimensional Skeleton Based Refinement (3D-SBR). In these schemes we take advantage of the lower dimensional character of the skeleton to simplify the description of the subdivision algorithm. As was stated there, an edge graph based structure is used to model longest edge bisection. For example, in the 2D case, using only the classification of the edges in a triangle as one longest edge and two other edges, an oriented graph is built. A similar graph is introduced in 3D to explain the different classes of possible tetrahedra in the subdivision phase. In [27] we presented an appropriate design for the data structure needed for the two dimensional skeleton-based refinement algorithm. Some properties showing the effectiveness of this design were also given. It was also shown that this design may be related to the concepts implicit in the Rivara *LEPP* (Longest Edge Propagation Path) algorithms [32]. As a result, finiteness of the 2D-SBR and of the 4-T scheme of Rivara is easily obtained.

Graph theory is particularly useful to model problems that emphasize the relationship among objects. In the present work we introduce the graph of the skeleton to explicitly derive data structures for the skeleton-based refinement algorithms. These data structures give a more natural and consistent approach for this class of refinement/coarsening algorithms.

The outline of this report is as follows: first some notation and definitions are introduced. Then, in Section 3 we present a brief review of the main refinement algorithms and associated partitions in 2D and 3D. The skeleton-based graph of an *n*-dimensional triangulation, for *n*=2,3 is developed in Section 4 and specific realizations of the graphs for longest edge bisection algorithms, such as the Skeleton Based Refinements algorithms in our work are given. Section 5 provides the new versions of the 2D and 3D Skeleton Based Refinement algorithms with the new approach using the skeleton graphs presented in the previous section. Finally, to illustrate the use and performance of the Skeleton Based Refinement algorithms, results from some experiments in 2D are given that focus on the data structures related to the skeleton graph.

2. DEFINITIONS

Definition 2.1 (*m*-simplex). Let $V=\{X_1,X_2,\ldots,X_{m+1}\}$ be a set of $m+1$ points in R^n, $m \le n$ such that $\{\overrightarrow{X_1X_i} : 2 \le i \le m+1\}$ is a linearly independent set of vectors. Then the closed convex hull of V denoted by $S =< V >=< X_1, X_2, \ldots X_{m+1} >$ is called an *m*-simplex in R^n, while the points $X_1, X_2, \ldots, X_{m+1}$ are called *vertices* of S, and the number m is said to be the *dimension* of S.

If $m = n$ a simplex S is simply called a *simplex* or *triangle* of R^n; 2- and 3-simplices are called triangles and tetrahedra as usual.

Definition 2.2 (*k*-face). Let $S = \langle X_1, X_2, \ldots, X_{n+1} \rangle$ be a *n*-simplex in R^n. A *k*-simplex $T = \langle Y_1, Y_2, \ldots, Y_k \rangle$ is called a *k*-subsimplex or a *k*-face of S if there are indices $0 \le i_0 < i_1 < \ldots < i_k \le n$ such that $Y_j = X_{i_j}$ for $0 \le j \le k$.

Obviously, the *0*- and *1*-faces of S are just its vertices and edges, respectively.

Definition 2.3 (triangulation). Let Ω be any bounded set in R^2, or R^3 with non-empty interior, $\overset{o}{\Omega} \neq \varnothing$, and polygonal boundary $\partial\Omega$, and consider a partition of Ω into a set of triangles $\tau = \{t_1, t_2, t_3, \ldots, t_n\}$. Then we say that τ is a triangulation if the following properties hold:

1. $\Omega = \cup t_i$
2. $\text{int}(t_i) \neq \varnothing, \forall t_i \in \Omega$
3. $\text{int}(t_i) \cap \text{int}(t_j) = \varnothing$, if $i \neq j$

Definition 2.4 (conforming triangulation). A triangulation τ of a bounded set Ω is called conforming (some authors prefer consistent, or compatible) if any pair of adjacent simplices share either an entire face or edge.

Definition 2.5 (skeleton). Let τ be any n-dimensional (n = 2 or 3) conforming triangular mesh. The k-skeleton of τ is the union of its *k*-faces. The $(n-1)$-skeleton is also called the skeleton.

For instance, the skeleton of a triangulation in three dimensions is comprised of the faces of the tetrahedra, and in two dimensions the skeleton is the set of the edges of the triangles. It should be noted however, that the skeleton can be understood as a new triangulation: if τ is a 3-D conforming triangulation in R^3, *skt(τ)* is a 2-D triangulation embedded in R^3. Furthermore, if τ is conforming, then *skt(τ)* is also conforming.

Note that if we define any simplex partition over a conforming triangulation in which some elements are divided, the application of such partition to a conforming triangulation always yields another (finer) triangulation.

In general, two (conforming) triangulations τ and τ^* of the same bounded set Ω are said to be *nested*, and we write $\tau \prec \tau^*$ if the following condition holds: $\forall t \in \tau$, $\exists t_1, t_2, \ldots, t_p \in \tau^*$ such that $t = t_1 \cup t_2 \ldots \cup t_p$. We also say that τ is coarser than τ^* or that τ^* is finer than τ.

Note that if over some initial mesh τ successive (local) refinements are performed, a sequence of nested triangulations is obtained. In a sequence of nested triangulations, the respective sets of nodes are also nested. This fact enables us to talk about *proper* nodes and *inherited* nodes as follows: let $T=\{\tau_1, \tau_2, \ldots, \tau_k\}$ be a sequence of nested grids, where τ_1 represents the initial mesh, τ_k the finest grid in the sequence. A node $N \in \tau_j$ is called a *proper node* of τ_j or a *j-new node* if N does not belong to any previous mesh. Otherwise, N is said to be an *inherited node* in τ_j. The edges, faces, and elements may be named similarly [23][30].

The *longest edge bisection* of a triangle t is the partition of the triangle by the midpoint of its longest edge and the opposite vertex. The *longest edge neighbor* of t is the

neighboring triangle t^* which shares with t the longest edge of t.

The *Longest Edge Propagation Path (LEPP)* of a triangle t_0 as defined by Rivara in [32] is an ordered list of all the adjacent triangles $\{t_0,t_1,...,t_n\}$ such that t_i is the neighbor triangle of t_{i-1} adjacent to the longest edge of t_{i-1}. It should be pointed out that when calculating the *LEPP* for a given triangle t, t also belongs to the *LEPP* list.

Next we provide some concepts and definitions from graph theory used later in this paper, [11][12].

A *geometric graph* G is a non-empty set of *nodes* P and a (possibly empty) set of *links* L. The graph is denoted as $G(P,L)$ or simply G. We denote $l_k \sim (p_i,p_j)$ to represent the link L_k associated with nodes p_i,p_j. Nodes usually represent objects and links relationships among objects. Links can have *labels* or *weights* and nodes have *names*. A subgraph $G(P',L')$ is a graph such that P' is contained in P and L' *is* contained in L. A *link* between nodes p_i and p_j denoted by (p_i, p_j) is said to be *incident* with the nodes p_i and p_j. The nodes p_i and p_j are called *endpoints* of the link (p_i, p_j). Two links are *adjacent* if they have a common node. The *degree* of a node p_i is the number of links incident in p_i. A *planar graph* is a graph which can be drawn in the plane such that links intercept, in a geometric sense, only at their nodes. A graph is *connected* if every pair of nodes in G is connected, otherwise it is said to be *disconnected*. A *component* is a maximally connected subgraph. If links on the graph are ordered then the graph is *directed* and if they are not it is *undirected*. For a given graph, if we can traverse from a node to a node on successively adjacent links without visiting the same links more than once, we call such a set of links a *chain* in an undirected graph and a *path* in a directed graph. The *length* of a path or a chain is equal to its number of links. If a chain or a path returns to the same node we call this a *circuit* and a *cycle* respectively. The *distance* between two nodes is the shortest path connecting these nodes. The traversal of a graph consists of visiting all its vertices. Depth-first traversal of a graph proceeds as follows: starting from an initial node, each adjacent first successor is visited, then each successor of this successor is visited and so on.

3. SOME REFINEMENT ALGORITHMS AND ASSOCIATED PARTITIONS

A number of adaptive mesh refinement algorithms have been developed in recent years. Considerable attention has been paid in particular to the refinement of simplicial grids. In this section we introduce some of the most common partitions for triangles and tetrahedra used in the literature. These partitions constitute the kernels of the related refinement algorithms.

3.1. Refinement algorithms and associated partitions in 2D

Definition 3.1 (4T Similar partition). The original triangle is divided into four triangles by connecting the midpoints of the edges by straight line segments. Consequently all the subtriangles are similar to the original one (see Figure 1.a). Carey, 1976 [10] was the first to apply this approach for Adaptive Mesh Refinement (AMR). This is one of the simplest partitions of triangles considered in the literature (see, for example, [1][3]). Bank *et al.* [1] have developed an algorithm based on this scheme: triangles initially marked for refinement are refined by using the 4T Similar Partition. This regular partition is called a *red partition*. To ensure the mesh conformity, a set of additional irregular refinement patterns (*green partition*) are performed by connecting mid-edge nodes to the apposite vertices of adjacent triangles.

Definition 3.2 (Single bisection and LE bisection). Single bisection consists of dividing the triangles into two sub-triangles by connecting the opposite vertex to the midpoint of one of the edges (see Figure 1.b). When the longest edge is chosen to bisect the triangle we say that a longest edge (LE) bisection has been done.

Definition 3.3 (4T-LE partition). The 4T Longest Edge partition bisects the triangle in four triangles as shown in Figure 1.c where the triangle is first subdivided by its longest edge, and then the two resulting triangles are bisected by joining the new midpoint of the longest edge to the midpoint of the remaining two edges of the original triangle.

Definition 3.4 (Baricentric partition). For any triangle t the baricentric partition of t is defined by connecting the vertices to the midpoints through the baricenter as shown in Figure 1d.

Rivara has described an effective algorithm for mesh refinement based on longest edge bisection [29][30]. A similar algorithm that deals with the skeleton is developed by Plaza and Carey in [25]. Both algorithms are equivalent in the sense that they lead to the same final triangulation.

Another approach by Mitchel is characterized as a newest-vertex-bisection scheme [20]. In this method the triangle edge opposite to the vertex that was most recently created is used for refinement. The associated partition is similar to that obtained from the 4T-Longest Edge (LE) scheme described later but with the two new bisections of the subtriangles on the initial longest edge.

3.2. Refinement algorithms and associated partitions in 3D

Several bisection techniques for tetrahedra have been developed in the last five years. Algorithms based on a *simple* longest edge bisection approach have been developed by Rivara and Levin [31] and by Muthukrishnan *et al.* [21]. Since each tetrahedron is divided by its longest edge until conformity is achieved, it is not known *a priori* into how many tetrahedra each of the original tetrahedra will be subdivided.

Recently Plaza and Carey [25] presented a generalization of the 4-T Rivara algorithm to three dimensions. This is termed the 3D Skeleton-Based Refinement algorithm. Some geometric properties of the algorithm have been pointed out by Rivara and Plaza [28]. The algorithm works first on the

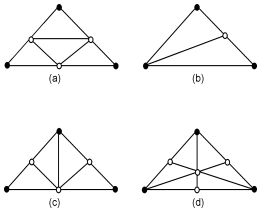


Fig. 1. Four partitions in 2D

triangular faces of the tetrahedra, the 2-skeleton of the 3D-tetrahedral tessellation, and then subdivides the interior of each tetrahedron in a manner consistent with the subdivision of the skeleton. This idea could be applied to develop similar algorithms in higher dimensions. The algorithm can be applied to any initial tetrahedral mesh without any preprocessing. This is an important feature with respect to other similar algorithms in the literature. As in the refinement case, the coarsening algorithm is based on derefining the 2-skeleton while ensuring the conformity of the mesh, and then reconstructing the interior of the tetrahedra. For this second task the former 3D (local) refinement patterns can be used in each tetrahedron.

The algorithms of Bänsch [2] and Liu and Joe [17] divide each tetrahedron into eight subtetrahedra by successive edge bisection. Kossaczký [15] has also proposed a recursive approach. His algorithm imposes certain restrictions and requires preprocessing of the initial mesh. The algorithm is equivalent, in the sense that it produces the same meshes as those of Bänsch, and Liu and Joe. Recently, Mukherjee [22] has also presented an algorithm equivalent to that of Bänsch and of Liu and Joe. However all these algorithms, although they produce the same meshes, are not applicable to any initial mesh and need some kind of pre-processing.

The analog of the 2D partition in Figure 1are given in Figures 2-4. In each case, on the left, we display a sample tetrahedron illustrating the partition used and on the right, a representation in which the faces have been cut open along the three edges which meet at vertex D and unfolded in a coplanar space R^2. Furthermore, to uniformly see the faces on the left side throughout this representation, a simple topological transformation is applied which leaves all the tetrahedron faces as equilateral. This idea of representing a tetrahedron has been used by Bänsch [2] to explain his refinement algorithm, and easily shows the graphical construction of the 3D partition from the point of view of subdividing the faces.

Definition 3.5 (Standard Partition). The original tetrahedron is divided into eight subtetrahedra by cutting off the four corners by planes trough the midpoints of the edges and subdividing the remaining octahedron by selecting one of three possible interior diagonals [4], Figure 2. This interior diagonal has to be chosen carefully in order to satisfy the non-degeneracy of the meshes obtained when the partition is recursively applied.

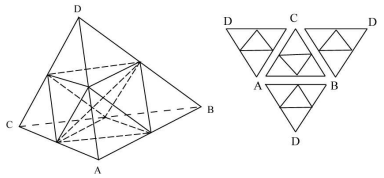


Fig. 2. 3-D Standard partition

Definition 3.6 (8T-LE partition). For any tetrahedron t with unique longest edge, the 8-Tetrahedra Longest Edge partition (8T-LE partition) of t is defined as follows:

1. LE-bisection of t producing tetrahedra t_1,t_2.
2. Bisection of t_i by the midpoint of the unique edge of t_i which is also the longest edge of a common face of t_i with the original tetrahedron t, producing tetrahedra t_{ij}, for $i,j=1,2$.
3. Bisection of each t_{ij} by the midpoint of the unique edge corresponding to an edge of the original tetrahedron.

In Figure 3 we show one of several different possible patterns that arise from the 8T-LE partition. In the Appendix we give all possible configurations that this partition produces.

Definition 3.7 (3-D Baricentric partition). For a tetrahedron t the baricentric partition is defined as follows:

1. Put a new node P at the baricentric point of t; put new nodes at the baricentric points of the faces of t; and put new nodes at the midpoints of the edges of t.
2. In each face of t construct the baricentric triangular partition of the face.
3. Join the baricentric point P with all the vertices of t, and with all the new nodes introduced as described above.

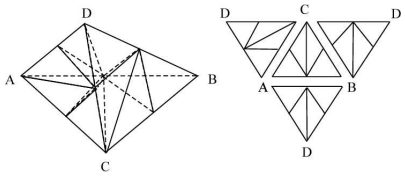


Fig. 3. Example of refinement pattern for the 8T-LE partition

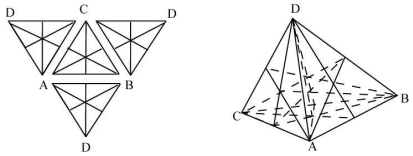


Fig. 4. 3-D Baricentric partition

At this point, two tools introduced before, the skeleton, (Definition 2.5) and the partition patterns of n-simplices, make it possible to outline a general framework for the Skeleton Based Refinement algorithms. Although our

specific version of these algorithms will be based on the 4T-LE partition for the two dimensional case and the 8T-LE for the three dimensional case, this general framework makes no assumptions on the specific partition, and it could be any of those described previously. The main data needed are the mesh to be treated, the *n*-simplex to be refined and the specific partition to use.

Our approach then proceeds to consider the successive skeletons in reverse order. Hence, the basic procedure for the *n*-D simplex consists of the *n* steps below:

```
Skeleton Based Refinement (mesh,n-simplex,partition)
For each n-simplex taking part in the refinement do
        1) Division of 1-skeleton
        2) Division of 2-skeleton
             . . .
        n) Reconstruction of the n-simplex
```

It should be noted here that step *n* of the previous algorithm concerning the reconstruction or appropriate subdivision of the *n*-simplex needs some geometric information (such as internal points, edges or faces and the connection among them) to perform the interior subdivision correctly. This information is specific to the particular partition and is intended to be provided by the general input parameter *partition*. In the next section we introduce the graph of the skeleton.

4. THE SKELETON BASED GRAPH OF A TRIANGULATION

From the skeleton concept induced by a simplex it is possible to build a graph that represents the topological adjacency among the *k*-faces in a mesh. For example, in Figure 5 we describe a hierarchical representation of the sets of skeletons for a tetrahedron that permits us to build up an associated graph. Here (a) is the reference tetrahedron in R^3, (Euclidean space); (b) shows the faces have been cut open along the three edges meeting at vertex D and unfolded in a coplanar space R^2 as before. As a result we obtain a representation of the 2-skeleton of the tetrahedron. In (c) the edges of the

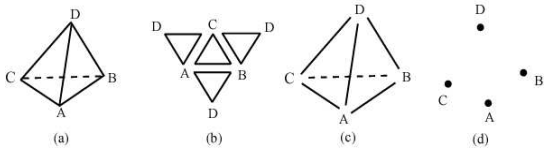


Fig. 5. Graphical representation of the skeleton sets

original tetrahedron are separated to form the 1-skeleton. Finally the four vertices define the 0-skeleton of the tetrahedron.

Definition 4.1 (*k-skeleton graph* $G^k(P,L)$). Let τ be an *n*-simplicial mesh with the corresponding *k*-skeleton sets, $k<n$. The *k*-skeleton undirected graph $G^k(P,L)$ of τ is built as follows. Let $P = \{p_0,p_1,...,p_r\}$ so that $p_i \in P$ is the node on the graph representing a *k*-face of τ. Let $L = \{l_0,l_1,...l_s\}$ so that $l_j \in L$ is the link on the graph representing a given relation *R* between two different nodes p_n, p_m of *P*. We denote $l_i \sim (p_m,p_n)$.

R represents the topological adjacency relationship between two *k*-faces of τ. Thus, for two different nodes p_n, p_m of *P* , $p_n R\ p_m$ iff the *k*-face represented by p_n shares a common *(k-1)*-face with the *k*-face represented by p_m.

For brevity we denote by $G^k(\tau)$ the *k*-skeleton graph. It can also be pointed out that, for a given *n*-simplicial mesh, the graph that refers to the simplices themselves is also of interest and used in Finite Element Mesh Generation [33]. Here we simply refer to it as the *n*-simplicial graph or $G(\tau)$. Figures 6.a.1 and 6.b.1 show the 1- and 2-skeleton undirected graphs of both a triangle and a tetrahedron.

One other useful feature that can be added to the graph as defined above is labels for links. For each link $l_i \in L$ and nodes p_n, $p_m \in P$ such that $l_i \sim (p_m,p_n)$ we assign a unique integer label that identifies the *(k+1)*-face or *n*-simplex in which the *k*-skeleton members represented by (p_m,p_n) are contained. Thus, one can recognize the upward relation of the *k*-skeleton members such as, edge->face, face->tetrahedron. This is very useful when programming the refinement algorithms. Furthermore, if it is relevant to know a relation such as edge->tetrahedron, the label will be a unique integer that refers to a given tetrahedron in the mesh. This feature is appropriate for the data structures of refinement/coarsening algorithms because it maintains upward links in the hierarchical representation of the skeleton.

Let us now consider as examples the triangle and tetrahedron and their associated 1- and 2-skeleton graphs. In Definition 4.1 the specification of the linkage of the nodes in the graph is determined by *R,* the natural adjacency relationship among the edges or faces in the *n*-simplex. In Figure 6 are shown the graph representations of not only that specific relation *R* but all possibilities for connecting edges and faces in 2D and 3D respectively. Different specifications of the graph are possible depending on the relations *R* between the *k*-faces.

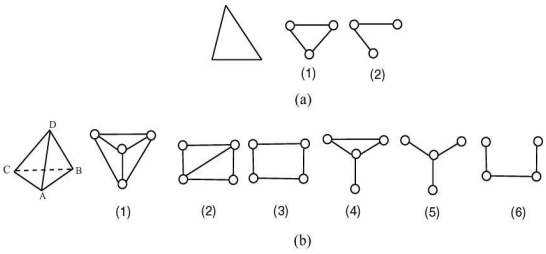


Fig. 6. Configurations for the *(n-1)*-skeleton graphs

To manage the configurations in Figure 6, we use an integer array T^{n-1}, $(n=2,3)$, which has as elements the different degrees of the nodes in the *(n-1)*-skeleton graph. The length of T^{n-1} is the number of elements in the *(n-1)*-skeleton set (3 for a triangle and 4 for a tetrahedron). Ordering in T^{n-1} is not relevant, so any permutation of array T^{n-1} is valid. For instance, for the graphs in Figures 6.a.1 and 6.b.4, we can use the arrays $T^1=[2,2,2]$ and $T^2=[2,2,3,1]$ respectively. The importance of introducing this array is that it defines *R* and hence it specifies the topological adjacency relationship desired in the mesh. For instance, $T^1=[2,2,2]$ and

T^2=*[3,3,3,3]* imply one will maintain all the possible topological relationship among the *k*-faces of the mesh, (each edge with the other two in a triangle and each face with the other three in the tetrahedron). However the adjacency defined by T^{n-1} may be restricted to achieve more efficient computation by, for example, minimizing the number of relations to be stored for a single *n*-simplex. Thus, the following minimizer is useful for this purpose,

$$T_n = \min_i \sum_j T_i^{n-1}(j)$$

where index *i* represents the different arrays T^{n-1} for a given *n* (*n*=2,3) and *j* indexes each element in the array.

T_n minimizes the number of links in the skeleton graph for the simplex. The configurations of G^{n-1} for T_n are shown in Table 1.

Table 1. Skeleton graphs and arrays T

Skeleton graph	*Array T*
	T^1=*[1,2,1]*
	T^2_1=*[3,1,1,1]*
	T^2_2=*[2,2,1,1]*

For our purpose of using the graph in modeling the refinement scheme based on longest edge bisection, we will restrict the relation *R* between the *k*-faces for the (*n-1*)-skeleton graph to those in Table 1. In the next section we justify this choice.

4.1 The 1-skeleton graph for longest edge bisection algorithms in 2D

Algorithms for 2D refinement based on the longest edge bisection of a triangle have the advantage that the different possibilities for refining a triangle depend on determining the longest edge. In general, we can say that these algorithms classify the edges of each triangle into two types: the longest edge named type 1, and the remaining two edges, named type 2, and this suffices for our purposes, (see Figure 7.a).

This idea can be expressed in terms of the 1-skeleton graph introduced previously by considering the relation *R*= "*length of edge x in the triangle is less than length of edge y*" and the array T^1 in Table 1. Because *R* is an order relation between edges, it is possible to add an orientation to the links in the graph that represent this ordering. Hence, the links in Figure 7.a indicate the dependency of the edges when refining to enforce conformity. In Figure 7.b a simple triangulation and the associated edge-based directed graph are shown.

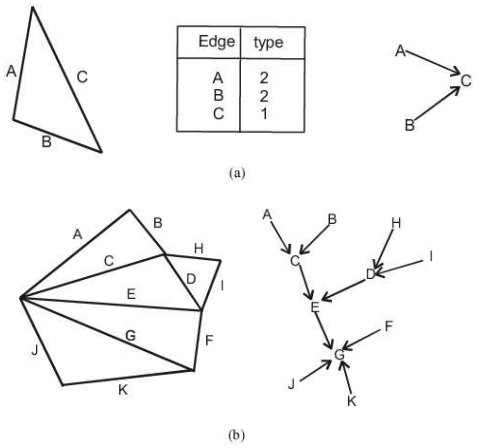


Fig. 7. 1-skeleton oriented graph in 2D

Proposition 1. *Let G^1 be the 1-skeleton directed graph of a two dimensional conforming triangulation τ. Then the graph does not contain cycles.*

Proof:
There are two situations in which loops (cycles) can occur. The first one arises in meshes having isosceles or equilateral triangles. In constructing or describing the graph the cases where a triangle in the *LEPP* is isosceles or equilateral may offer more than one longest edge for bisection. In these situations where more than one such edge can be chosen we select the first one as arbitrarily assigned in setting up the edge data. The second situation is as described in Figure 8. Let $X=\{x_0, x_1, x_2, x_3, x_n\}$ be the common edges shared by the adjacent triangles in the *LEPP* of t_0 (t_0 is the triangle defined by edges *c* and x_0). If a cycle occurs then $length(c) \leq length(x_0) \leq \ldots\ length(x_n) \leq length(c)$. However, this is impossible when the edges in *X* are distinct. If the edges in *X* are equal, we are in the first situation considered in this proof.

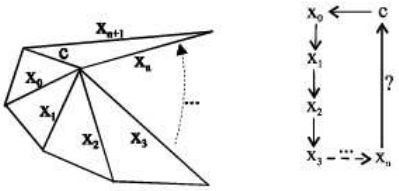


Fig. 8. Triangulation and simplified associated graph

From the computational point of view, the next proposition ensures the locality of the 1-skeleton directed graph data structure.

Proposition 2. *The cost for updating the 1-skeleton directed graph data structure when applied the 4T-LE bisection affects at most three nodes in each refinement step.*

Proposition 3.- *Let τ be a two-dimensional conforming triangulation with N triangles and t be an arbitrary triangle of τ. Let e be the longest edge of t. Then the LEPP of t can be obtained from a depth-first traversal of the 1-skeleton directed graph of τ starting from the node e of the graph, with a maximum cost of order O(N-1).*

4.2 The 1- and 2-skeleton graph for longest edge bisection algorithms in 3D

We first consider the 1-skeleton graph of a tetrahedron. As in 2D, the same criterion for the relationship between edges is applied. In this case, we can distinguish three different types of edges according to the local length in a single tetrahedron (similar to Bänsch [2], Liu and Joe [17] and Mukherjee [22]). The longest edge of a tetrahedron is denoted type 1. The longest edge of each one of the other two faces is type 2, and the rest of the edges are type 3.

Remark 3: By small (hypothetical) perturbation of the coordinates of the nodes one can ensure that there is exactly one longest edge. The longest edge of a tetrahedron is frequently called the *reference edge.*

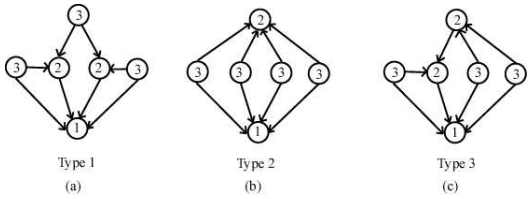


Fig. 10. The 1-skeleton oriented graph for the three types of tetrahedra

For the 8T-LE partition, we distinguish three types of tetrahedra based on the relative position of their edge-types [25]. First, we consider tetrahedra obtained by bisecting the type 1 edge, second by the edge or edges type 2 and so on. Each type of tetrahedron can be represented by a 1-skeleton oriented graph as in Figure 10. In the Appendix the possible configurations of the 8T-LE partition and the associated 1-skeleton oriented graphs are given.

Remark 2: Some further remarks about the 1-skeleton directed graph for a tetrahedron are warranted:

1. The 1-skeleton directed graph has exactly 4 nodes and 8 links;
2. The degree of the node representing the longest edge is 4;
3. The 1-skeleton directed graph is a planar and disconnected graph and does not contain cycles;
4. Each path is of length 1.

The 1-skeleton directed graph of a tetrahedron as presented above is an edge-based representation that can be used to design the data structure of any other longest edge bisection scheme for tetrahedra (see section 3.2). (It provides the information needed to perform the refinement of a tetrahedron in terms of the edges.)

From the computational point of view, one can design the refinement algorithm to construct the graph locally for each tetrahedron dynamically. This is the approach we use here for memory efficiency. Alternatively one could construct the graph and store it for every tetrahedron in the mesh and then update the graph as refinement occurs. This implies an extra data storage of *8n, n* being the number of tetrahedra in the mesh.

In [24] asymptotic results were discussed concerning the adjacency relations between the topological elements in a 3D mesh when applying the 8T-LE partition. For instance, each edge is shared by 36/7 tetrahedra on average. Taking this into account, in the 1-skeleton graph, a node could have on average up to (36/7)*4 links. This would imply an unacceptable storage requirement, and for this reason it is not efficient to maintain the 1-skeleton graph globally connected in the mesh but locally for each tetrahedron. Next, we will explore the 2-skeleton graph as a suitable data structure to extend globally to all the mesh.

Remark 4: The representation in Figures 11.a-b is unique. In this case edge *AB* is the longest edge. This representation is called the *standard position* [2]. We call the faces *f1* and *f4 reference faces*.

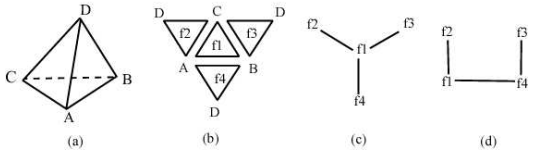


Fig. 11. The 2-skeleton graph in 3D

We have two alternatives to build the 2-skeleton graph for a single tetrahedron (Figure 11c,d). Using the array notation introduced at the beginning of Section 4, the graph in Figure 11.c is denoted by $T^2_1=[3,1,1,1]$. In this case the node with degree 3 is one of the two reference faces (if *AB* is the longest edge in the tetrahedron, then the reference faces are *f1* and *f4*). The second possibility is shown in Figure 11.d and denoted by $T^2_2=[2,2,1,1]$. Then each node with degree 2 is intended to represent a reference face (sharing the reference edge of the tetrahedron).

We construct the 2-skeleton undirected graph, $G^2(\tau)$ for a 3-D triangulation τ, emphasizing that two possible local configurations for the faces of a single tetrahedron can be assumed. The first one, is depicted in Figure 11.c and the second one, in Figure 11.d. Both configurations are identical for our purposes, having the same properties for the final graph. By constructing such a graph, we can derive a face-based data structure that suits the need of the 3D-SBR algorithm as explained in the next section.

Proposition 4. *Let $G^2(\tau)$ be the 2-skeleton undirected graph of a three dimensional triangulation τ, then the graph is planar and connected.*

5. THE 2D-3D SKELETON BASED REFINEMENT ALGORITHMS

In this section we present the 2D and 3D Skeleton Based Refinement algorithms taking into account the graph based data structure introduced previously. It should be pointed out here that the approach of Skeleton Based Refinement in 2D has also been used by the MatLab Partial Differential Equation Toolbox to implement mesh refinement. This toolbox uses a scheme for refinement which is a combination of regular and single longest edge partition.

The partitions used for refining the triangulation in our algorithms are the 4T-LE approach for the 2D case and the 8T-LE approach for 3D. With this skeleton graph the algorithms admit, a more natural and consistent description.

The 2D-SBR algorithm as presented by Plaza and Carey [25] basically performs the refinement task by using two sequential steps as follows: identify and bisect the edges specified by the overall refinement indicators, (steps 1 and 2 below), and subdivide the individual triangles to define the new triangulation, (step 3 below).

```
Algorithm 2D-SBR(τ, t0)
/* Input: τ mesh, t0 triangle
/* Output: τ mesh
1.L=1-skeleton(t0)
For each edge ei ∈ L do
        Subdivision(ei)
End
2.For each edge ei ∈ L do
        Si=LEPP(ei,G1(τ))
        For each triangle ti ∈ Si do
                Let ei be the longest edge of ti
                Subdivision(ei)
        End
End
3.For each triangle tj ∈ τ to be subdivided do
        Subdivision(tj)
End
```

Subdivision is a procedure that subdivides either an edge or a triangle. The subdivision of an edge consists of the bisection of the edge at its midpoint. The subdivision of a triangle depends upon the number of bisected edges. If it has one bisected edge, then the single longest edge bisection is performed (Definition 3.2). If it has two bisected edges, first the longest edge bisection is made and then the midpoints of the two bisected edges are joined. If, finally, it has three edges bisected, we proceed as in the previous case and add a new connection from the midpoint of the longest edge to the third bisected edge (the 4T-LE partition Definition 3.3).

Concerning computational time, it should be noted that the 2D-SBR algorithm has two cost-intensive parts but is of linear complexity with respect to the number of nodes. The first one arises from the creation and updating of the data and the cost depends on the coarseness of the initial mesh. Subsequent updating of the data structure is local to a triangle as stated in Proposition 2. The second point corresponds to step 2 for identification and bisection of the edges involved throughout the overall refinement process. The worst case occurs if the initial triangulation is such that each triangle in the path has its longest edge longer than the previous one. However, numerical experiments carried out in this report reveal that this cost asymptotically approaches a small constant bound as such a mesh is refined.

In the case of the 3D-SBR algorithm, the four major steps are. (1) Subdivision of edges that form the 1-skeleton of the input tetrahedron; (2) Subdivision of edges due to the propagation of the conformity; (3) Having bisected the edges implied in the refinement, subdivide the corresponding faces according to the 4T-LE partition for 2D; (4) The last step performs the appropriate interior subdivision of the tetrahedron taking part in the refinement, following the 8T-LE partition strategy.

The 3D-SBR algorithm then follows as:

```
Algorithm 3D-SBR(τ,t0)
/* Input: τ mesh, t0 tetrahedron
/* Output: τ mesh
L=1-skeleton(t0)
1:For each edge ej ∈ L do
        Subdivision(ej)
End
2.While L<>∅ do
    Let ek be an element ∈ L
    For each tetrahedron ti ∈ hull(ek) do
        For each non-conforming face fi ∈ G2(ti) do
                Let ep be the longest edge of fi
                Subdivision(ep)
                L=L∪ep
        End
  End
    L=L-ek
End
3.For each face fi ∈ G2(τ) to be subdivided do
        Subdivision(fi)
End
4.For each tetrahedron ti ∈ τ to be subdivided do
        Subdivision(ti)
End
```

It should be pointed out here that the procedure *subdivision* subdivides the successive skeletons in *reverse* order: steps 1 and 2 perform the subdivision of the edges, step 3 performs the subdivision of the faces and step 4 performs the interior subdivision to subtetrahedra.

The algorithm is of linear complexity in the number of nodes, as stated in [25]. The points where the 2-skeleton graph $G^2(\tau)$ is used are clearly seen in the previous algorithm. First, the inner loop of step 2 accesses the non-conforming faces in the mesh and in step 3 the subdivision of the 2-skeleton is performed. In both steps, the 2-skeleton graph provides access to the faces taking part in the refinement in a local and efficient way.

In Step 2, the procedure *hull(e)* provides the set of simplices $S \in \tau$ such that $e \in S$. That is, *hull(e)* is comprised of all the simplices that share the same edge *e*. In [25] an algorithm for finding the *hull* of an edge is given and is of linear complexity.

The previous versions of the 2D/3D-SBR algorithm perform refinement of single elements. However, the algorithms can obviously be used for refinement of a collection of elements, by simply providing this collection as input instead a single element.

6. EXPERIMENTS ON THE SKELETON-BASED ALGORITHM (2D CASE)

In [26] we describe an application of the 3D-SBR algorithm combined with a derefinement algorithm.

In the following discussion of numerical results we consider a particular domain corresponding to the Gran Canaria Island (Spain) and local refinement using 4T longest edge bisection on three contiguous subdomains. The graph data structure has been implemented and demonstrated in the numerical experiments. We focus on the following three points: (1) a demonstration of the skeleton based algorithm as described in the preceding sections of this paper; (2) a numerical timing study for this test problem showing the variation in the total CPU time as the number of nodes increases with each mesh refinement and 3) some statistical measures of the *LEPP* subgraph determined from the graph at each refinement step.

As indicated above, in this first study we consider the Gran Canaria domain with mesh refinement on three subgrids. The island domain is subdivided into three contiguous subdomains S1, S2 and S3, corresponding to areas in which the elevation of the terrain is progressively greater. The goal is to successively refine the three subdomains according to the three elevation regions so defined.

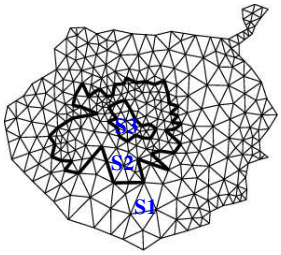


Fig. 12. Gran Canaria Mesh showing the three subdomains S3, S2, S1

Six levels of refinement are performed as follows: from the initial mesh (level 1, 592 elements) in Figure 12, which is a Delaunay mesh, the innermost subdomain S3 is first chosen and all its elements are refined to obtain the mesh level 2. Note that the refinement propagates outside S3 to ensure conformity of the triangulation as one would expect. This level 2 mesh for the entire domain contains 736 elements. In the second step of this demonstration the new subgrid on S3 is chosen and these elements are refined with propagation outside S3 as before to obtain mesh level 3 with 1230 elements. In the third step, we elect to refine all elements in S2 in a similar sense and obtain mesh level 4 with 2624 elements. In the fourth step, we choose to refine all elements in (S1∪S2) and obtain mesh level 5 with 9258. Next, we elect to refine all elements in subgrid (S2∪S3) and obtain mesh level 6 with 30730 elements. Finally, the elements in subgrid S1 are chosen and we obtain mesh level 7 with 41448 elements.

The successive meshes are obtained by refinement using the *LEPP* approach and the 2D-SBR algorithm as described earlier. These results effectively demonstrate the use of the graph based data structure for SBR.

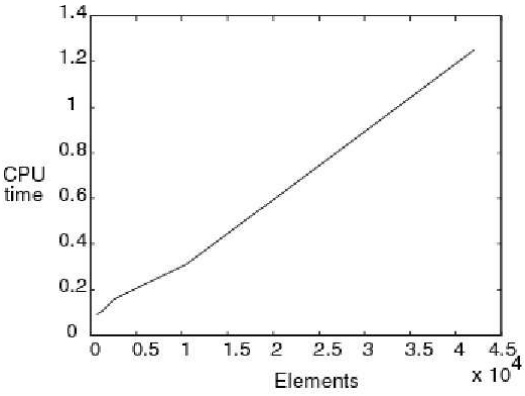
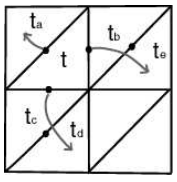


Fig. 13. CPU time for the test problem

In the next part of the study we plot the CPU time against the number of nodes for each of the meshes obtained using the above refinement strategy. The linear complexity of both the algorithm for refinement and also the graph based data structure are demonstrated by the linear growth plot in Figure 13.

Finally, in the third part of the study we briefly report on the behavior of the *LEPP* statistics for the SBR approach. As noted in Proposition 3 earlier, the *LEPP* is computed directly from the subgraph of our graph based data structure. We report results using two metrics. The first metric is the sum of the lengths of *LEPP's* of the neighbors of a triangle *t,* the number of additional triangles refined when refining *t*. The second metric is the maximum length of *LEPP's* of the neighbors of *t*. We will refer to these metrics as M1 and M2 respectively. For instance, M1=5 and M2=2 when refining *t* as in Figure 14, and they are typical values for practical cases.



Triangle	*LEPP*
t	$\{ t, t_a \}$
t_a	$\{t_a\}$
t_b	$\{ t_b, t_e \}$
t_c	$\{ t_c, t_d \}$

Fig. 14. *LEPP* when refining a triangle

In Tables 2 and 3, the mean, standard deviation and maximum values of M1 and M2 are given.

The results in Table 2 show that M1, the number of additional elements obtained when refining a single triangle, on average tends to 5. This mean that the 1-skeleton graph as presented in section 4.1 is on average traversed for approximately 5 nodes per element. The column Max M1 reveals the maximum value of M1 for the elements in the mesh at each level of refinement. This number is reduced by

45% from the initial mesh (level 1) to the last mesh (level 7). These results as well as those in the previous experiments show that the cost of performing the local refinement per element with the 2D-SBR algorithm decreases as the number of refined elements increases.

Table 2. M1 statistics for Gran Canaria Mesh

Level	*N. Elem.*	*Mean M1*	*Std M1*	*Max M1*
1	592	6.6199	2.7647	20
2	736	6.6902	2.5771	20
3	1230	6.5057	2.2658	20
4	2624	6.0191	1.6793	15
5	9258	5.5134	1.1629	11
6	30730	5.2478	0.6814	11
7	41448	5.2128	0.5642	10

Table 3. M2 statistics for Gran Canaria Mesh

Level	*N. Elem.*	*Mean M2*	*Std M2*	*Max M2*
1	592	3.4510	1.6600	10
2	736	3.5394	1.6631	10
3	1230	3.3837	1.6017	9
4	2624	2.9184	1.0519	8
5	9258	2.4265	0.6017	7
6	30730	2.3672	0.5090	7
7	41448	2.1891	0.3287	7

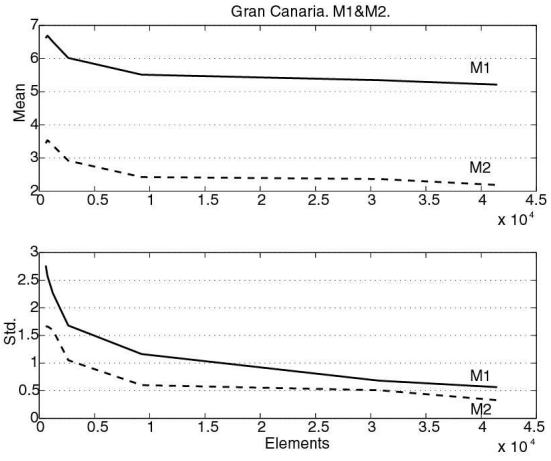


Fig. 16. Statistics for Gran Canaria Mesh

The results in Table 3 concern the metric M2: The mean value of the longest edge propagation path for the elements tends to 2 and the maximum of this number is decreased by 30% from the initial mesh to the final one. In terms of the 1-skeleton graph, this implies that the cost for obtaining the *LEPP* when refining a single element, on average tends to 2.

The performance of the *LEPP* approach using these metrics is the subject of considerable interest since the utility of the longest edge bisection schemes and their efficient use of memory may be in question. This approach to study the SBR scheme using the *LEPP* metrics will be presented in more depth in a following study.

7.CONCLUSIONS AND FUTURE WORK

We have reviewed several refinement ideas in both two and three dimensions and extended our previous work on skeleton-based refinement algorithms. For the 2D-SBR algorithm, an improved and very natural edge-data structure based on graphs has been introduced and studied. Several properties of the supporting data structure are described, showing this to be efficient in storage and time cost. The extension of the graph based data structure for the 3D case it is also presented, and implies a corresponding face-based data structure.

We discuss in detail the graphs for 2D and 3Drefinement of triangulations. Finally, numerical experiments to illustrate the ideas are presented using the 2D-SBR algorithm.

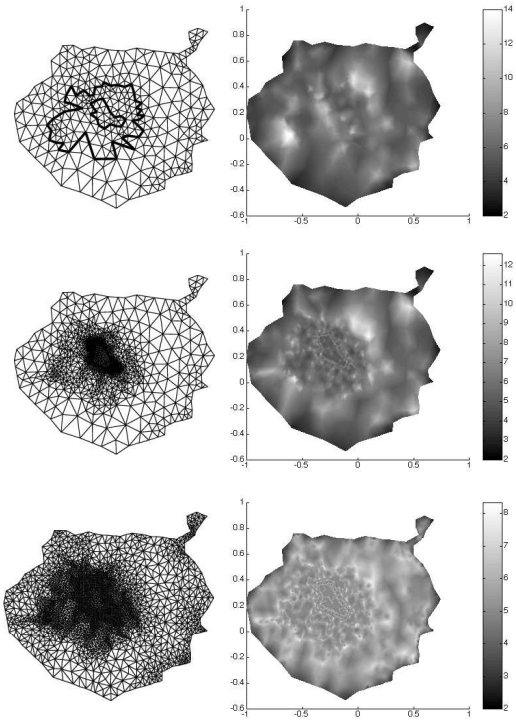


Fig 17. Gran Canaria meshes corresponding to refinement levels 1, 4, 5 and associated spatial distribution of metric M1

ACKNOWLEDGEMENT

This work has been supported in part by Project number PI-1999/146 from Gobierno de Canarias and by ASCI grant number B347883.

APPENDIX. TETRAHEDRAL 1-SKELETON ORIENTED GRAPH CONFIGURATIONS

In the figure below we describe the possible configurations for the 1-skeleton oriented graph of the three different types of tetrahedra. On the left, an unfolded reference tetrahedron is shown and on the right, its corresponding graph. Edge *e1* represents the *reference edge* as stated in Remark 3 and for each face of the tetrahedron the longest edge is indicated by a dashed line.

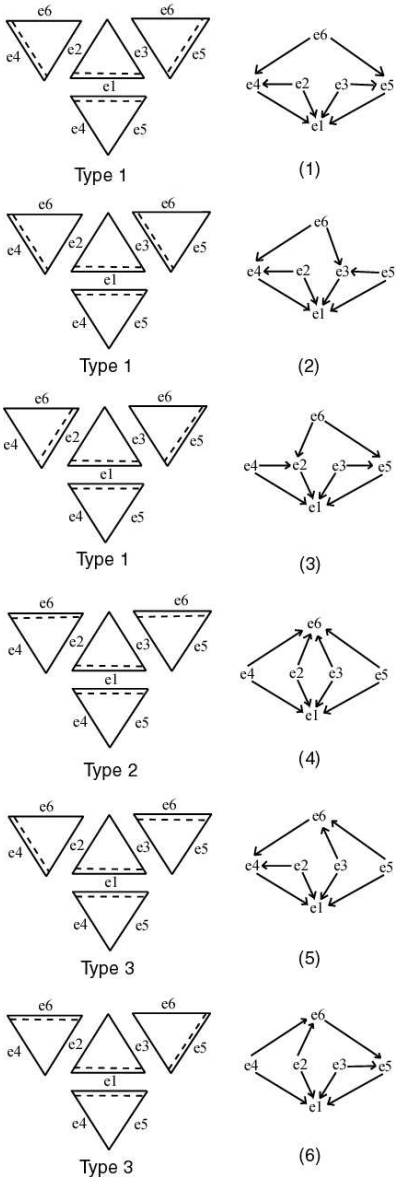


Fig. 17. Configurations of G^1 for the different tetrahedra

REFERENCES

[1] R. E. Bank, and Andrew H. Sherman, "The use of adaptive grid refinement for badly behaved elliptic differential equations", in R. Vichnevetsky and R.S. Stepleman, Eds., *Advances in Computer Methods for Partial Differential Equations III*, pp.33-39, (1979).

[2] E. Bänsch, "Local mesh refinement in 2 and 3 dimensions", *IMPACT Com. Sci. Eng.,* 3, pp.181-191 (1991)

[3] E. B. Becker, Graham F. Carey, and J. Tinsley Oden, "Finite elements", *Prentice-Hall*, (1981).

[4] J. Bey, "Simplicial grid refinement: on Freudenthal´s algorithm and the optimal number of congruence classes", *Numer. Math*., (2000). URL:http://dx.doi.org/10.1007/5002111111108

[5] J. Bonet and J. Peraire, "An alternating digital tree (ADT) algorithm for 3D geometric and intersection problems", *Int. J. Numer. Methods Eng*., 31, pp.1-17 (1991).

[6] G.F. Carey, "Computational Grids: generation, adaptation and solution strategies", *Taylor & Francis*, (1997).

[7] G.F. Carey, and A. Pehlivanov, "An adaptive octree-based scheme for hierarchic extraction compression and remote visualization of data", *Fifth USNCCM, Boulder, CO, Aug 4-6,* (1999) *(to appear in TICAM report 2000 series; to be submitted to IJNME 2000).*

[8] G.F. Carey, A. I. Pehlivanov, R. C. Mastroleo, R. McLay, Y. Shen, V. Carey and R. Dutton, "Hierarchic visualization and parallel computation", *High performance computing ' 98 Proceedings, Boston, MA, April* (1998).

[9] G.F. Carey, M. Sharma and K.C. Wang, "A class of data structures for 2-D and 3-D adaptive mesh refinement", *Int. J. Numer. Methods Eng*., 26, pp. 2607-2622 (1988).

[10] G.F. Carey, "A mesh refinement scheme for Finite Element computations*", J. Comput. Meth. Appl. Mech. Eng*., 7, 1, pp. 93-105, (1976).

[11] J.L. Gross, T.W. Tucker, "Topological Graph Theory", *J. Wiley & Sons*, (1987).

[12] F. Haray, "Graph Theory", *Addison Wesley*, (1972).

[13] Y. Kallinderis and P. Vijayan, "Adaptive refinement-coarsening scheme for three-dimensional unstructured meshes", *AIAA J*., 31, pp.1440-1447 (1993).

[14] L. Kettner, "Using generic programming for designing a data structure for polyhedral surfaces", *Comp. Geom.-Theory and Applications*, 13, pp.65-90, (1999).

[15] I. Kossaczký, "A Recursive Approach to Local Mesh Refinement in Two and Three Dimensions", *J. Comp. App. Math.,* 55, pp.275-288 (1994).

[16] P. Leinen, "Data structures and concepts for adaptive finite element methods", *Computing* 55, pp.325-354 (1995).

[17] A. Liu, and B. Joe, "Quality Local Refinement of Tetrahedral Meshes based on Bisection", SIAM. *J. Sci. Statist. Comput.,* 16 pp.1269-1291 (1995).

[18] R. Löhner, "Some useful data structures for the generation of unstructured grids", *Comm. Appl. Numer. Methods*, 4, 123-135 (1988).

[19] R. Löhner, "Edges, star, super-edges and chains", *Comput. Methods Appl. Mech. Eng*., 111, pp.255-263 (1994).

[20] W.F. Mitchel, "Adaptive refinement for arbitrary finite element spaces with hierarchical basis", *J. Comput. Appl. Math.*, 36, pp.65-78, (1991).

[21] E. Muthukrishnan, P.S. Shiakolas, R.V. Nambiar, and K.L. Lawrence. "Simple algorithm for adaptive refinement of three-dimensional finite element Tetrahedral Meshes", *AIAA Journal,* 33, pp.928-932 (1995).

[22] A. Mukherjee, "An adaptive finite element code for elliptic boundary value problems in three dimensions with applications in numerical relativity", *Ph. D. Thesis,* Penn. State University, (1996).

[23] A. Plaza, L. Ferragut, and R. Montenegro, "Derefinement algorithms of nested meshes", in J. Van Leeuwen, ed. "Algorithms, software, architecture", pp.409-415, *Elsevier Sci. Pub.,* (1992).

[24] A. Plaza, "Asymptotic behavior of the average of the adjacencies of the topological entities in some simplex partitions", *in 8th Inter. Mesh. Roundtable, October 10-13, South Lake Tahoe, California,*(1999).

[25] A. Plaza, and G. F. Carey, "Local refinement of simplicial grids based on the skeleton", *App. Num. Math.,* 32, pp.195-218 (2000).

[26] A. Plaza, M. A. Padrón, and G. F. Carey, "A 3D refinement/derefinement algorithm for solving evolution problems", *App. Num. Math.,*32, pp.401-418 (2000).

[27] A. Plaza, J. P. Suárez, M. A. Padrón, "Mesh graph structure for longest-edge refinement algorithms", *Sandia Report SAND 98-2250*, pp. 335-344, (1997).

[28] M.C. Rivara, and A. Plaza, "Mesh selective refinement /derefinement based on the 8-tetrahedra longest-edge partition", *Dept. of Computer Science, U. de Chile, TR/DCC-99-6-1999,*(1999).

[29] M.C. Rivara, "Selective refinement/derefinement algorithms for sequences of nested triangulations", *Int. J. Num. Meth. Eng.,* 28, pp.2889-2906, (1989).

[30] M.C. Rivara, "Mesh refinement based on the generalized bisection of simplices", *SIAM J. Numer. Anal.*, 2, pp.604-613 (1984).

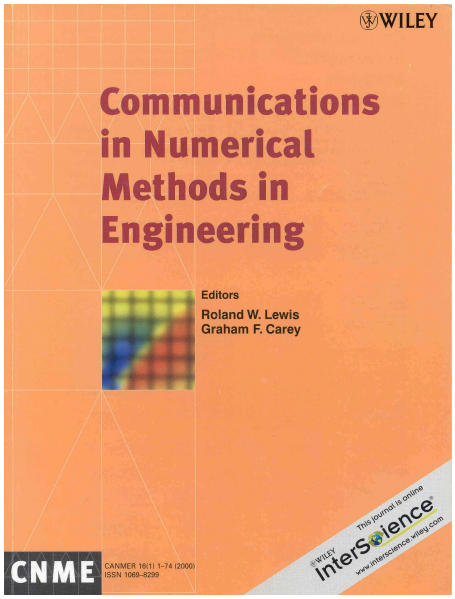
[31] M.C. Rivara and C. Levin, "A 3-D refinement algorithm suitable for adaptive and multi-grid techniques", *Comm. in App. Num. Meth*., 8, 281-290, (1992).

[32] M.C. Rivara, "New mathematical tools and techniques for the refinement and/or improvement of unstructured triangulations", *5th. International Meshing Roundtable'96, Sandia Report SAND96-2301,UC405*, 77-86, (1996).

[33] L.T. Souza and M. Gattass, "A new scheme for mesh generation and mesh refinement using graph theory", *Computers&Structures, 46,6,1073-1084,* (1993).

[34] K. Weiler, "Edge-based data structures and concepts for solid modeling in curved-surface environments", *IEEE 0272-1716/85/0100-0021*, (1985).

A.3. Communications in Numerical Methods in Engineering



COMMUNICATIONS IN NUMERICAL METHODS IN ENGINEERING
Commun. Numer. Meth. Engng 2000; **00**:1–6 *Prepared using* cnmauth.cls *[Version: 2000/03/22 v1.0]*

Graph Based Data Structures for Skeleton Based Refinement Algorithms

J. P. Suárez[1,*] G.F. Carey[2] and A. Plaza[1]

[1] *University of Las Palmas de Gran Canaria, Canary Islands, Spain, josepa@cicei.ulpgc.es*
[2] *University of Texas at Austin, Austin, Texas, U.S.A., carey@cfdlab.ae.utexas.edu*

SUMMARY

In this paper we discuss a class of adaptive refinement algorithms for generating unstructured meshes in two and three dimensions. We focus on Skeleton Based Refinement (SBR) algorithms as proposed by Plaza and Carey [10] and provide an extension that involves the introduction of the graph of the skeleton for meshes consisting of simplex cells. By the use of data structures derived from the graph of the skeleton, we reformulate the Skeleton Based Refinement scheme and devise a more natural and consistent approach for this class of adaptive refinement algorithms. As an illustrative case, we discuss in detail the graphs for 2D refinement of triangulations and for 3D we propose a corresponding new face-based data structure for tetrahedra. Experiments using the two dimensional algorithm and exploring the properties of the associated graph are provided. Copyright © 2000 John Wiley & Sons, Ltd.

KEY WORDS: mesh refinement; skeleton; data structures; edge bisection

1. INTRODUCTION

The skeleton view of a mesh provides a hierarchical set of simplex components defining the mesh in a recursive manner as the dimension is decreased. Thus, a mesh of tetrahedra can be defined using a skeleton made up of the surface triangles in the tessellation, these triangles can be viewed as composed of a skeleton of edges, which in turn are composed of a skeleton of nodes. In [10] Plaza and Carey first developed algorithms for grid refinement based on the skeleton in two and three dimensions.

In 2D, the longest edge bisection of a triangle t is the partition of the triangle by the midpoint of its longest edge and the opposite vertex. The longest edge neighbor of t is the neighboring

*Correspondence to: Dto. Cartografía y Expresión Gráfica en la Ingeniería. University of Las Palmas de Gran Canaria, Canary Islands, Spain.

Contract/grant sponsor: Gobierno de Canarias; contract/grant number: PI-1999/146

Contract/grant sponsor: DOE ASCI; contract/grant number: B347883

Contract/grant sponsor: DOD; contract/grant number: NRC-CR-97-0002

Received
Revised

triangle t^* which shares with t the longest edge of t. The Longest Edge Propagation Path ($LEPP$) of a triangle t as defined by Rivara in [12] is an ordered list of all the adjacent triangles $\{t_0 = t, t_1, \ldots, t_n\}$ such that t_i is the neighbor triangle of t_{i-1} adjacent to the longest edge of t_{i-1}. The $4T$ Longest Edge ($4T$-LE) partition bisects the triangle in four triangles where the triangle is first subdivided by its longest edge, and then the two resulting triangles are bisected by joining the new midpoint of the longest edge to the midpoint of the remaining two edges of the original triangle.

In 3D, for any tetrahedron t with unique longest edge, the 8-Tetrahedra Longest Edge ($8T$-LE) partition of t is defined as follows: (1) LE-bisection of t producing tetrahedra t_1, t_2; (2) Bisection of t_i by the midpoint of the unique edge of t_i which is also the longest edge of a common face of t_i with the original tetrahedron t, producing tetrahedra t_{ij}, for $i, j = 1, 2$; and (3) Bisection of each t_{ij} by the midpoint of the unique edge corresponding to an edge of the original tetrahedron.

We use standard notations and terminology in graph theory as in [5][6]. We denote $l_k \sim (p_i, p_j)$ to represent the link l_k associated with nodes p_i, p_j. A planar graph is a graph which can be drawn in the plane such that links intercept, in a geometric sense, only at their nodes. The depth-first traversal of a graph implies complete traversal of the graph as follows: starting with an initial node, a successor is visited, then the successor of this successor, and so on, until no further successor is found. Then the next successor of the initial node is visited, applying the same process again.

The outline of this paper is as follows: Section 2 develops the skeleton-based graphs of an n-dimensional triangulation, for $n = 2, 3$. Section 3 provides the new versions of the 2D and 3D Skeleton Based Refinement algorithms using the skeleton graph approach. Finally, results from some experiments in 2D are given to illustrate the use of the algorithm and data structure.

2. THE SKELETON BASED GRAPH OF A TRIANGULATION

Definition 1 (k-skeleton graph $G^k(P, L)$) *Let τ be an n-simplicial mesh with the corresponding hierarchy of k-skeleton sets, $k < n$. The k-skeleton undirected graph $G^k(P, L)$ of τ is built as follows: Let $P = \{p_0, p_1, \ldots, p_r\}$ so that p_i is a k-face of τ, and $L = \{l_0, l_1, \ldots, l_s\}$ so that l_j represents the topological adjacency relationship $\mathbf{R}$ between two different nodes of P, that is between two k-faces of τ.*

Thus, for two different nodes p_n, p_m of P, $p_n \mathbf{R} p_m$ iff the k-face represented by p_n shares a common $(k-1)$-face with the k-face represented by p_m. For brevity we denote by $G^k(\tau)$ the k-skeleton graph. For each link $l_i \in L$ and nodes $p_n, p_m \in P$ such that $l_i \sim (p_m, p_n)$ we can assign a unique integer label that identifies the $(k+1)$-face or n-simplex in which the k-skeleton members represented by (p_m, p_n) are contained. Thus, one can recognize the upward relation of the k-skeleton members such as, edge-face, face-tetrahedron. Furthermore, if it is relevant to know a relation such as edge-tetrahedron, the label will be a unique integer that refers to a given tetrahedron in the mesh. This feature is appropriate for the data structures of refinement/coarsening algorithms because it maintains upward links in the hierarchical representation of the skeleton.

2.1. The 1-skeleton graph for longest edge bisection algorithms in 2D

Algorithms for 2D refinement based on the longest edge bisection of a triangle may use this property to develop directed graphs for the data structure. For example, in the presente work, we classify the edges of each triangle into two types: the longest edge named type 1, and the remaining two edges, named type 2, and this idea can be expressed in terms of the 1-skeleton graph introduced previously by considering the relation $\mathbf{R}$= *"length of edge x in the triangle is less than length of edge y"*. Because $\mathbf{R}$ is an order relation between edges, it is possible to add an orientation to the links in the graph that represent this ordering. Hence, the links in Figure 1 indicate the dependency of the edges when refining to enforce conformity.

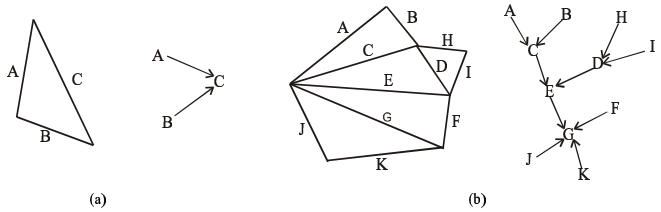


Figure 1. 1-skeleton oriented graph in 2D.

Proposition 1. *Let G^1 be the 1-skeleton directed graph of a two dimensional conforming triangulation τ. Then the graph does not contain cycles [11].*

Proposition 2. *Updating the 1-skeleton directed graph data structure by applying 4T-LE bisection affects only up to three graph nodes in each refinement step.*

From the computational point of view, the previous proposition ensures the locality of the 1-skeleton directed graph data structure.

Proposition 3. *Let τ be a two-dimensional conforming triangulation with N triangles and t be an arbitrary triangle of τ. Let e be the longest edge of t. Then the $LEPP$ of t can be obtained from a depth-first traversal of the 1-skeleton directed graph of τ starting from the node e of the graph, with a maximum cost of order $O(N-1)$.*

Numerical experiments carried out in this report reveal that asymptotically, the average cost per triangle of traversing the $LEPP$, approaches a small constant.

2.2. The 1- and 2-skeleton graph for longest edge bisection algorithms in 3D

We first consider the 1-skeleton graph of a tetrahedron. As in 2D, the same criterion for the relationship between edges is applied. In this case, we can distinguish three different types of edges: The longest edge of a tetrahedron is denoted type 1; The longest edge of each of the other two faces not sharing this type 1 edge is type 2, and the remaining edges are type 3.

Remark 1. *By small (hypothetical) perturbation of the coordinates of the nodes one can ensure that there is exactly one longest edge. The longest edge of a tetrahedron is frequently called the reference edge.*

Remark 2. *Some comments concerning the 1-skeleton directed graph for a tetrahedron are warranted:*

1. The 1-skeleton directed graph has exactly 4 nodes and 8 links;

2. The degree of the node representing the longest edge is 4;

3. The 1-skeleton directed graph is a planar and disconnected graph and does not contain cycles;

4. Each path is of length 1.

The 1-skeleton directed graph of a tetrahedron is an edge-based representation that can be used to model the data structure of any longest edge bisection scheme for tetrahedra (see Section 2.1). From the computational point of view, one can design the refinement algorithm to construct the graph locally for each tetrahedron dynamically. This is the approach we use here for memory efficiency. Alternatively one could construct the graph and store it for every tetrahedron in the mesh and then update the graph as refinement occurs. This implies an extra data storage of $8n$, n being the number of tetrahedra in the mesh.

When applying the 8T-LE partition, each edge is shared globally by 36/7 tetrahedra on average [9]. Therefore, in the 1-skeleton graph a node could have on average up to $(36/7) \times 4$ links. This would imply an unacceptable storage requirement, and for this reason it is not efficient to maintain the 1-skeleton graph globally. Instead we provide it locally for each tetrahedron.

Remark 3. *The representation in Figures 2.a-b is unique. In this case edge AB is the longest edge. This representation is called the standard position. We call the faces $f1$ and $f4$ reference faces.*

We have two alternatives to build the 2-skeleton graph for a single tetrahedron. The first one is the graph in Figure 2.c. In this case the node with degree 3 is one of the two reference faces. The second possibility is shown in Figure 2.d. Then each node with degree 2 is intended to represent a reference face. Both configurations could be used for our purposes. By constructing such a graph, we derive a face-based data structure that suits the need of the 3D-SBR algorithm as explained in the next section.

Proposition 4. *Let $G^2(\tau)$ be the 2-skeleton undirected graph of a three dimensional triangulation τ, then the graph is planar and connected.*

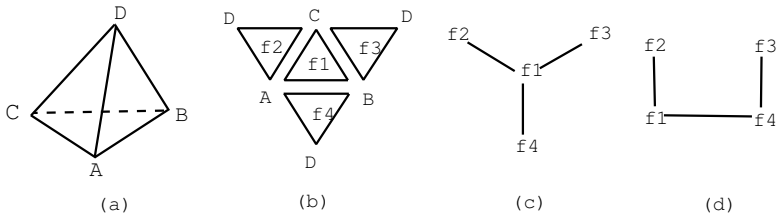


Figure 2. The 2-skeleton graph in 3D.

3. THE 2D-3D SKELETON BASED REFINEMENT ALGORITHMS

The partitions used for refining the triangulation in our algorithms are the 4T-LE approach for the 2D case and the 8T-LE approach for 3D. With the skeleton graph, the algorithms admit a more natural and consistent description (see Figure 3).

The 2D-SBR algorithm as presented by Plaza and Carey [10] works in two sequential stages as follows: (i) identify and bisect the edges specified by the overall refinement indicators, (steps 1 and 2 in Figure 3), and (ii) subdivide the individual triangles to define the new triangulation, (step 3). Concerning computational time, it should be noted that the 2D-SBR algorithm is of linear complexity with respect to the number of nodes. Numerical experiments carried out in this report reveal that this cost approaches asymptotically a small constant as such a mesh is refined.

Algorithm 2D-SBR(τ, t_0)
/* Input: τ mesh, t_0 triangle
/* Output: τ mesh
1. L=1-skeleton(t_0)
For each edge $e_i \in$ L **do**
Subdivision(e_i)
End
2.For each edge $e_i \in$ L **do**
S_i=LEPP(e_i,$G^1(\tau)$)
For each triangle $t_i \in S_i$ **do**
Let e_i be the longest edge of t_i
Subdivision(e_i)
End
End
3.For each triangle $t_j \in \tau$ to be subdivided **do**
Subdivision(t_j)
End

Algorithm 3D-SBR(τ,t_0)
/* Input: τ mesh, t_0 tetrahedron
/* Output: τ mesh
1. L=1-skeleton(t_0)
For each edge $e_j \in$ L **do**
Subdivision(e_j)
End
2.While L $\neq \varnothing$ **do**
Let e_k be an element $\in$ L
For each tetrahedron $t_i \in$ hull(e_k) **do**
For each non-conforming face $f_i \in G^2(t_i)$ **do**
Let e_p be the longest edge of f_i
Subdivision(e_p)
L=L$\cup e_p$
End
End
L=L-e_k
End
3.For each face $f_i \in G^2(\tau)$ to be subdivided **do**
Subdivision(f_i)
End
4.For each tetrahedron $t_i \in \tau$ to be subdivided **do**
Subdivision(t_i)
End

Figure 3. The 2D-SBR and 3D-SBR Algorithms.

In the case of the 3D-SBR algorithm, the four major steps are: (1) Subdivision of edges that form the 1-skeleton of the input tetrahedron; (2) Subdivision of edges due to the propagation of the conformity to the neighbor tetrahedra; (3) Having bisected the edges implied in the refinement, subdivide the corresponding faces according to the 4T-LE partition for 2D; (4) Perform the appropriate interior subdivision of the tetrahedron taking part in the refinement, following the 8T-LE partition strategy. It should be pointed out here that the procedure *subdivision* in Figure 3 subdivides the successive skeletons in reverse order: Steps 1 and 2 perform the subdivision of the edges, step 3 performs the subdivision of the faces and step 4

performs the subdivision of tetrahedra. The algorithm is of linear complexity in the number of nodes, as stated in [10]. The points where the 2-skeleton graph $G^2(\tau)$ is used are clearly seen in the previous algorithm. First, the inner loop of step 2 accesses the non-conforming faces in the mesh and in step 3 the subdivision of the 2-skeleton is performed. In both steps, the 2-skeleton graph provides access to the faces taking part in the refinement in a local and efficient way. In step 2, the procedure $hull(e)$ provides the set of simplices $S \in \tau$ such that $e \in S$. That is, $hull(e)$ is comprised of all the simplices that share the same edge e.

4. EXPERIMENTS ON THE SKELETON-BASED ALGORITHM (2D CASE)

We consider a particular domain corresponding to the Gran Canaria Island (Spain) and local refinement using 4T longest edge bisection on three contiguous subdomains. We focus on the following two points: (1) A demonstration of the skeleton based algorithm as described in the preceding sections of this paper; (2) Some statistical measures of the $LEPP$ subgraph determined at each refinement step. In the first study we consider the Gran Canaria domain S with mesh refinement on disjoint subregions S_1, S_2 and S_3 with $S = S_1 \cup S_2 \cup S_3$, for innermost region S_3, intermediate region S_2 and outermost annular region S_1. The goal is to successively refine the three subdomains according to the three elevation regions so defined. The sequences of meshes are obtained by refinement using the $LEPP$ approach and the 2D-SBR algorithm as described earlier. These results effectively demonstrate the use of the graph based data structure for SBR.

In the second part of the study we briefly report on the behavior of the $LEPP$ statistics for the SBR approach. As noted in Proposition 3 earlier, the $LEPP$ is computed directly from the subgraph in our graph based data structure. We report results using two metrics. The first metric is the number of additional triangles refined when refining t. The second metric is the maximum length of $LEPP$'s of the neighbors of t without counting triangle t. We will refer to these metrics as $M1$ and $M2$ respectively.

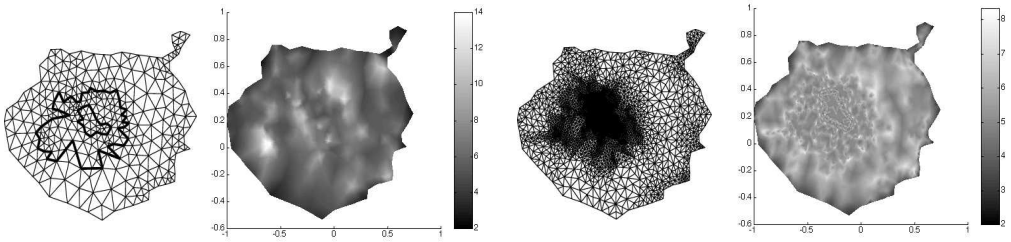


Figure 4. Gran Canaria meshes corresponding to refinement levels 1 and 5, and associated spatial distribution of metric $M1$.

In Figure 5, the mean and standard deviation of $M1$ and $M2$ are given. The results show that $M1$, on average tends to 5. This means that the 1-skeleton graph as presented in section 2.1 is on average traversed for 5 nodes per element. These results as well as those in the previous experiments show that the cost of performing the local refinement per element with the 2D-

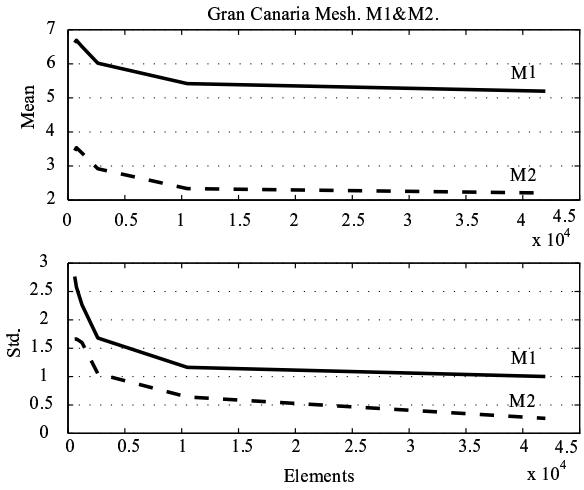


Figure 5. Statistics for Gran Canaria mesh.

SBR algorithm approaches a fixed acceptable value as the number of refinement elements increases. It also suggests that a demanding refinement criterion can be used (since there will be moderate additional neighbor subdivision).

This result for the $LEPP$ approach using these metrics is of considerable interest sincethe utility of this type of longest edge bisection scheme and their efficient use of memory has previously been questioned. Further details will be presented in more depth in a following study.

5. CONCLUSIONS

We have reviewed several refinement ideas in both two and three dimensions and extended our previous work on skeleton-based refinement algorithms. For the 2D-SBR algorithm, an improved and very natural edge-data structure based on graphs has been introduced and studied. Several properties of the supporting data structure are described, showing this to be efficient in storage and time cost. The extension of the graph based data structure for the 3D case it is also presented, proposing a corresponding face-based data structure. We discuss in detail the graphs for 2D refinement of triangulations and for 3D. Finally, numerical experiments to illustrate the ideas presented in the paper are presented using the 2D-SBR algorithm.

ACKNOWLEDGEMENTS

This research has been supported in part by Gobierno de Canarias, grant Number PI1999/146, by DOE ASCI Contract No. B347883, and by DOD Contract No. NRC-CR-97-0002.

REFERENCES

1. Bonet J., Peraire J. An alternating digital tree (ADT) algorithm for 3D geometric and intersection problems. *Int. J. Numer. Methods Eng.* 1991; **31**:1–17.
2. Carey G.F. *Computational Grids: generation, adaptation and solution strategies.* Taylor & Francis, 1997.
3. Carey G.F. A mesh refinement scheme for finite element computations. *J. of Comp. Meth. App. Mech. Eng.* 1976; **7**(1):93–105.
4. Carey G.F., Sharma M., Wang K.C. A class of data structures for 2-D and 3-D adaptive mesh refinement. *Int. J. Numer. Methods Eng.* 1988; **26**:2607–2622.
5. Gross J.L., Tucker T.W. *Topological Graph Theory.* J. Wiley & Sons, 1987.
6. Haray F. *Graph Theory.* Addison Wesley, 1972.
7. Kettner L. Using generic programming for designing a data structure for polyhedral surfaces. *Comp. Geom. Theory and Applications* 1999; **13**:65–90.
8. Leinen P. Data structures and concepts for adaptive finite element methods. *Computing* 1995; **55**:325–354.
9. Plaza A. Asymptotic behavior of the average of the adjacencies of the topological entities in some simplex partitions. *SANDIA REPORT SAND 99-2288* 1999; 233–240.
10. Plaza A., Carey G.F. Local refinement of simplicial grids based on the skeleton. *App. Num. Math.* 2000; **32**:195–218.
11. Suárez J.P., Carey G.F., Plaza A., Padrón M.A. Graph based data structures for skeleton based refinement algorithms. *TICAM REPORT series* 2001; **01-10**.
12. Rivara M.C. Mesh refinement based on the generalized bisection of simplices. *SIAM J. Numer. Anal.* 1984; **2**:604–613.

Bibliografía

[1] A.V. Aho, J. E. Hopcroft and J.D. Ullman, *Data structures and algorithms*, Addison Wesley, 1983.

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

[3] I. Babuška and W.C. Rheinboldt, *Error estimates for adaptive finite element computations*, SIAM J. Numer. Anal., 15 (1978), pp. 736–754.

[4] I. Babuška and W.C. Rheinboldt, *A posteriori error analysis of finite element solutions for one-dimensional problems*, SIAM J. Numer. Anal., 18 (1981), pp. 565–589.

[5] R.E. Bank and A.H. Sherman, *The use of adaptive grid refinement for badly behaved elliptic partial differential equations*, in Advances in Computer Methods for Partial Differential Equations III, R. Vichnevetsky and R.S. Stepleman, eds, IMACS, 1979, pp. 33–39.

[6] R.E. Bank and A.H. Sherman and H. Weiser, *Refinement algorithms and data structures for regular local mesh refinement*, R. Stepleman et al., Scientific Computing, North-Holland, 1983, pp. 3–17.

[7] E. Bänsch, *Local mesh refinement in 2 and 3 dimensions*, IMPACT Com. Sci. Eng., 3 (1991), pp. 181–191.

[8] B.G. Baumgart, *A polyedron representation for computer vision*, Proc. AFIPS Natl. Comput. Conf., 44 (1975), pp. 589–596.

[9] M. Berger, *Geometry*, Springer-Verlag, 1987.

[10] J. Bey, *Tetrahedral grid refinement*, Computing, 55 (1995), pp. 355–378.

[11] J. Bonet and J. Peraire, *An alternate digital tree algorithm for geometric searching and intersection problems*, Int. J. Num. Meth. Eng., 31 (1991), pp. 1–17.

[12] G.F. Carey, *A mesh refinement scheme for finite element computations*, Com. Math. App. Mech. Eng., 7, 1 (1976), pp. 93–105.

[13] G.F. Carey, M. Sharma and K.C. Wang, *A class of data structures for 2-d and 3-d adaptive mesh refinement*, Int. J. Num. Meth. in Eng., 26 (1988), pp. 2607-2622.

[14] P.G. Ciarlet, *The finite element method for elliptic problems*, in Studies in mathematics and its applications, J. Lion *et al.*, Eds., North-Holland, 1978.

[15] P. Cignoni, C. Montani and R. Scopigno, *A comparison of mesh simplification algorithms*, Comp. & Graph., 22, 1 (1998), pp. 37-54.

[16] T.J. Cross, *A two-dimensional triangular mesh generator using the advanced front method*, Dept. Civil Engineering, U. Swansea U.K., Technical Report CR/662/91.

[17] L. De Floriani, P. Magillo and E. Puppo, *Applications of computational geometry to geographic information systems*, in Handbook of Computational Geometry, J.-R. Sack and J. Urrutia, Eds., Elsevier, 2000, pp. 333-388.

[18] L. De Floriani, B. Falcidieno, G. Nagy and C. Pienovi, *A hierarchical structure for surface approximation*, Comp. & Graph., 8, 2 (1984), pp. 183–193.

[19] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, 1987.

[20] J.M. Escobar, *Generación de mallas tridimensionales mediante triangulación de Delaunay*, Tesis Doctoral, Universidad de Las Palmas de Gran Canaria, 1995.

[21] L. Ferragut, NEPTUNO, *Un código para el método de elementos finitos adaptativo*, (en FORTRAN) Dept. Mat. Aplic. y Met. Inf., ETSI Minas, Madrid, 1987.

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

[23] L. Ferragut, R. Montenegro, G. Winter and A. Nuñez, *Accurate extraction of interconnect capacitances by adaptative mixed finite element method*, Proceedings of the EUROMICRO 91, North–Holland (1991), pp. 61–68.

[24] S. Fortune, *Voronoi Diagrams and Delaunay Triangulations*, in Computing in Euclidean Geometry, pp. 193–233, World Cientific, Singapore, 1992.

[25] W.H. Frey, and D.A. Field, *Mesh Relaxation: A new technique for improving triangulations*, Int. J. Num. Meth. in Eng., 31 (1991), pp. 1121-1133.

[26] J.P.S.R. Gago, D.W. Kelly, and O.C. Zienkiewicz, *A posteriori error analysis and adaptive processes in the finite element method. Part II: Adaptive mesh refinement*, Int. J. Num. Meth. Eng., 19 (1983) pp. 1621–1656.

[27] H. George, *Automatic mesh generation*, J. Wiley, 1991.

[28] M.T. Goodrich and K. Ramaiyer , *Geometric data structures* , in Handbook of Computational Geometry, J.-R. Sack and J. Urrutia, Eds., Elsevier, 2000, pp. 463-489.

[29] J.L. Gross, T.W. Tucker, *Topological Graph Theory*, John Wiley and Sons, 1987.

[30] L. Guibas, J. Stolfi, *Primitives for the manipulation of general subdivisions and the coomputation of Voronoi Diagrams*, ACM Transaction on Graphics, 4(3) (1985) pp. 74–123.

[31] P.L. George, F. Hecht and E. Saltel, *Automatic mesh generator with specified Boundary*, Comp. Meth. in Mech. and Eng., 92 (1991) pp. 259–268.

[32] R.L. Graham, D.E. Knuth, and O. Patashnik, *Concrete Mathematics*, John Wiley and Sons, 1989.

[33] W. Hackbush and V. Trottengurg (eds), *Multigrid Methods II, Lectures Notes in Mathematics*, Springer–Verlag, 1982, Berlin.

[34] W. Hackbush and V. Trottengurg (eds), *Multigrid Methods II, Lectures Notes in Mathematics*, Springer–Verlag, 1986, Berlin.

[35] J. Hartmann and J. Wernecke, *The VRML 2.0 Handbook*, Addison-Wesley, 1998.

[36] D.M. Hawken, P. Townsend and F. Webster, *The use of dynamic data structures in finite element applications*, Int. J. Num. Meth. Eng., 33 (1992), pp. 1795–1811.

[37] C.M. Hoffmann, *How to construct the skeleton of CSG objects*, in 4th IMA Conference on Mathematics of Surfaces, Bath, England, (1990).

[38] C.M. Hoffmann, *Geometric and solid modeling*, Morgan Kaufmann Publishers, Inc., 1989.

[39] H. Hoppe, *Efficient implementation of progressive meshes*, Comp. & Graph., 22, 1 (1998), pp. 27–36.

[40] R. Hoppe, *Bemerkung der Redaktion*, Archiv Math. Physik (Grunert), 56, pp. 307-312, 1874.

[41] H. Jin and R. I. Tanner, *Generation of unstructured tetrahedral meshes by the advancing front technique*, Int. J. Num. Meth. Eng., 36 (1993), pp. 1805–1823.

[42] B. Johnston, J. Sullivan and A. Kwasnike, *Automatic conversion of triangular finite element meshes to quadrilateral elements*, Int. J. Numer. Meth. Eng., 31 (1991), pp. 67–84.

[43] B. Kearfott, *A proof of convergence and an error bound for the method of bisection in* $\mathbb{R}^n$, Math. Comp., 32 (1978), pp. 1147–1153.

[44] L. Kettner, *Using generic programming for designing a data structure for poliedral surfaces*, Comp. Geometry, 13 (1999), pp. 65–90.

[45] I. Kossaczký, *A recursive approach to local mesh refinement in two and three dimensions*, J. Comp. App. Math., 55 (1994), pp. 275–288.

[46] P. Lindstrom, D. Koller, W. Ribarsky, L. Hodges, N. Faust and G. Turner, *Real-time, continuous level of detail rendering of height fields*, Procceedings of SIGGRAPH'96, (1996).

[47] A. Liu and B. Joe, *On the shape of tetrahedra from bisection*, Math. Comp., 63 (1994), pp. 141–154.

[48] A. Liu and B. Joe, *Quality local refinement of tetrahedral meshes based on bisection*, SIAM J. Sci. Comput., 16 (1995), pp. 1269–1291.

[49] S.H. Lo, *A new mesh generation scheme for arbitrary planar domains*, Int. J. Num. Meth. Eng., 21 (1985), pp. 1403–1426.

[50] S.H. Lo, *Perspective projection of non–convex polyhedra*, Int. J. Num. Meth. Eng., 26 (1988), pp. 1485–1506.

[51] R. Lohner and P. Parikh, *Three–dimensional grid generation by the advancing front method*, Int. J. Num. Meth. Fluids, 8 (1988), pp. 1135–1149.

[52] The Mathworks Inc., *Partial Differential Equations (PDE) Toolbox User's Guide* , Matlab Online manuals.

[53] J.M. Maubach, *Local bisection refinement for n-simplicial grids generated by reflection*, SIAM J. Sci. Stat. Comp., 16 (1995), pp. 210–227.

[54] W.F. Mitchell, *Optimal multilevel iterative methods for adaptive grids*, SIAM J. Sci. Statist. Comput. 13 (1992), pp. 146–167.

[55] M. Mantyla, *An introduction to Solid Modeling*, Computer Press, 1988.

[56] J.R. Munkres, *Elementary differential topology*, Princeton Univ. Press, second edition, 1966.

[57] A. Mukherjee, *An adaptive finite element code for elliptic boundary value problems in three dimensions with applications in numerical relativity*, PhD. Thesis, Penn. State University, University Park, PA 16802, 1996.

[58] D.E. Muller and F.P. Preparata, *Finding the intersection of two complex polyedra*, Theoret. Comput. Sci., 7 (1978), pp. 217-236.

[59] E. Muthukrishnan, P.S. Shiakolas, R.V. Nambiar, and K.L. Lawrence, *Simple algorithm for adaptive refinement of three-dimensional finite element tetrahedral meshes*, AIAA Journal, 33 (1995), pp. 928–932.

[60] K. Nakahashi and D. Sharov, *Direct surface triangulation using the advancing front method*, AIAA–95–1686–CP, (1995).

[61] J. Nievergelt and P. Widmayer, *Spatial data structures: concepts and design choices*, in Handbook of Computational Geometry, J.-R. Sack and J. Urrutia, Eds., Elsevier, 2000, pp. 725-764.

[62] J. Pach (Ed.), *New trends in discrete and computational geometry*, Springer-Verlag, 1993.

[63] M.A. Padrón, *Un algoritmo de desrefinamiento en dimensión tres para mallas encajadas de tetraedros basado en el esqueleto*, Tesis Doctoral, Universidad de Las Palmas de Gran Canaria, 1999.

[64] R. Pajarola, *Large scale terrain visualization using the restricted quadtree triangulation*, in Proc. IEEE Visualization'98, 1998, pp. 19–26 & p. 515.

[65] J. Peraire, M. Vahdati, K. Morgan and O.C. Zienkiewicz, *Adaptive remeshing for compressible flow computations*, J. Comp. Phys., 72 (1987), pp. 449–466.

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

[67] A. Plaza, *Sobre el número de triángulos generados por la partición 4T-LE*, en preparación.

[68] A. Plaza, and G.F. Carey, *A new refinement algorithm for tetrahedral grids based on skeleton*, TICAM Technical Report 96-54, November 1996.

[69] A. Plaza and G.F. Carey, *Local refinement of simplicial grids based on the skeleton*, TICAM Technical Report, 98-03, January 1998.

[70] A. Plaza, J.P. Suárez and M.A. Padrón, *Mesh graph structure for longest-edge refinement algorithms*, in Proceedings 7th International Meshing Roundtable'98, October, 1998 Michigan, SANDIA Report SAND 98-2250, pp. 335-344.

[71] A. Plaza and G.F. Carey, *Local refinement of simplicial grids based on the skeleton*, Appl. Num. Math., 32/2 (2000), pp. 195–218.

[72] A. Plaza, M.A. Padrón and G.F. Carey, *A 3D refinement/derefinement algorithm for solving evolution problems*, Appl. Num. Math., 32/4 (2000), pp. 401–418.

[73] A. Plaza y M.A. Padrón, *Un algoritmo de desrefinamiento en 3D para mallas de tetraedros basado en el esqueleto*, Rev. Int. Met. Num. Cal. Dis. Ing., 15, 4 (1999), pp. 471–483.

[74] A. Plaza and M.C. Rivara, *El comportamiento asintótico de la media del grado de los nodos para mallas de símplices*, en 8 Encuentros de Geometría Computacional, Castellón, 1999.

[75] A. Plaza and M.C. Rivara *On the adjacencies of triangular meshes based on skeleton-regular partitions*, J. Comp. and App. Math., por aparecer (2002).

[76] F.P. Preparata and M.I. Shamos *Computational Geometry: and introduction* Springer-Verlag Berlin, 1985.

[77] M. Price, C. Stops and G. Butlin, *A medial object toolkit for meshing and other applications*, in 4th International Meshing Roundtable, Albuquerque, New Mexico, (1995).

[78] W.C. Rheinboldt and C.K. Mesztenyi, *On a data structure for adaptative finite element mesh refinements*, ACM Transactions on Mathematical Software, 6 (1980), pp. 166–187.

[79] M.C. Rivara, *Mesh refinement based on the generalized bisection of simplices*, SIAM J. Numer. Anal., 2 (1984), pp. 604–613.

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

[81] M.C. Rivara, *Selective refinement/derefinement algorithms for sequences nested triangulations*, Int. J. Num. Meth. Eng., 28 (1989), pp. 2889–2906.

[82] M.C. Rivara, *Local modification of meshes for adaptive and/or multigrid finite-element methods*, J. Comp. and Appl. Math., 36 (1991), pp. 79–89.

[83] M.C. Rivara and C. Levin, *A 3-d refinement algorithm suitable for adaptive and multi-grid techniques*, J. Comp. and Appl. Math., 8 (1992), pp. 281–290.

[84] M.C. Rivara, *New mathematical tools and techniques for the refinement and/or improvement of unstructured triangulations*, in Proceedings 5th International Meshing Roundtable'96, October 10-11, 1996 Pittsburgh, SANDIA Report SAND 96-2301, UC-405, pp. 77–86.

[85] M.C. Rivara and M. Venere, *Cost analysis of the longest-side triangle bisection*, Eng. Comp., 12 (1996), pp. 224–234.

[86] M.C. Rivara, *New longest-edge algorithms for the refinement and/or improvement of unstructured triangulations*, Int. J. Num. Meth. Eng., 40 (1997), pp. 3313–3324.

[87] M.C. Rivara and G. Iribarren, *The 4-triangles longest-side partition of triangles and linear refinement algorithms*, Math. Comp., 65, 216 (1997), pp. 1485–1502.

[88] M.C. Rivara and A. Plaza, *Mesh refinement/derefinement based on the 8-tetrahedra longest-edge partition*, Dept. of Computer Science, U. de Chile, TR/DCC-99-6-1999, (1999).

[89] I.G. Rosenberg and F. Stenger, *A lower bound on the angles of triangles constructed by bisecting the longest side*, Math. Comp., 29 (1975), pp 390–395.

[90] D. Salomon, *Computer graphics & Geometric Modeling*, Springer, 1999.

[91] O. Santana y M. Díaz, *Estructuras de datos multidimensionales*, Facultad de Informática, Universidad de Las Palmas de Gran Canaria, 1996.

[92] M.S. Shephard, F. Guerinoni, J.E. Flaherty, R.A. Ludwig and P.L. Baehman, *Finite octree mesh generation for automated adaptive 3D flow analysis*, in Numerical grid generation in computational fluid mechanics'88, Miami, 1988.

[93] K. Shimada, and D.C. Gossard, *Bubble mesh: Automated triangular meshing of non-manifold geometry by sphere packing*, in Third Symp. on Solid Modeling and Appls., 1992, pp. 409–419.

[94] J.P. Suárez, G.F. Carey, A. Plaza, and M.A. Padrón, *Graph based data structures for skeleton based refinement algorithms*, TICAM Technical Report, 01-10, Abril, 2001.

[95] J.P. Suárez, G.F. Carey and A. Plaza, *Graph based data structures for skeleton based refinement algorithms*, Comm. Num. Meth. Eng., por aparecer (2002).

[96] M. Stynes, *On faster convergence of the bisection method for all triangles*, Math. Comp., 35 (1980), pp. 1995–1201.

[97] K. Weiler, *Edge-based data structures and concepts for solid modeling in curved-surface environments*, IEEE Computer Graphics and Applications, 5, 1, January (1985), pp. 21-40.

[98] J. Wernicke, *The Inventor Mentor: Programming Object-Oriented 3D Graphics with Open Inventor*, Release 2, Addison Wesley, 1994.

[99] D.F. Watson, *Computing the n-dimensional Delaunay tessellation with aplication to Voronoi polytopes*, The Computer Journal, 24 (1981), pp. 167–172.

[100] N.P. Weatherhill and O. Hassan, *Efficient three dimensional grid generation using the Delaunay triangulation*, Com. Flu. Dynam., 2 (1992), pp. 961–968.

[101] H.S. Wilf, *Generatingfunctionology*, Academic Press, 1994.

[102] M.A. Yerry and M.S. Shephard, *A modified-quadtree approach to finite element mesh generation*, IEEE Com. Graphics Appl., 3, 1 (1983), pp. 39–46.

[103] M.A. Yerry and M.S. Shephard, *Automatic three-dimensional mesh generation by the modified-octree technic*, Int. J. Num. Meth. Eng., 20 (1984), pp. 1965–1990.

[104] J.Z. Zhu and O.C. Zienkiewicz, *Adaptive techniques in the finite element method*, Int. J. Num. Meth. Eng., 32 (1991), pp. 783–810.

[105] O.C. Zienkiewicz and J.Z. Zhu, *A simple error estimator and adaptive procedure for practical engineering analysis*, Int. J. Num. Meth. Eng., 24 (1987), pp. 337–357.

[106] O.C. Zienkiewicz and J.Z. Zhu, *Adaptivity and mesh generation*, Int. J. Num. Meth. Eng., 32 (1991), pp. 783–810.