Acceso y actualización de una red irregular de triángulos desde un quadtree

Antonio Ruiz
Ismael Colomina
Institut Cartogràfic de Catalunya
Balmes, 209-211. 08006-Barcelona
E-mail: toni@icc.es, ismael@icc.es

Lluís Pérez
Universitat Politècnica de Catalunya
Dep. de Llenguatges i Sistemes Informàtics
Secció d'Informàtica Gràfica
Av. Diagonal 647, 08028-Barcelona
E-mail: lpv@lsi.upc.es

Resumen

Se presentan dos estructuras dinámicas de datos que, conjuntamente, permiten el acceso rápido y la actualización de una red irregular de triángulos (TIN) de gran volumen. Las estructuras presentadas son variantes de la lista doblemente conexa de segmentos (DCEL) y del *bucket point region quadtree (bucket PR-quadtree)*.

1 Introducción

1.1 Motivación y requerimientos

En el Institut Cartogràfic de Catalunya se planteó la necesidad de construir una base de datos para mantener el modelo altimétrico de Cataluña. El problema ha sido dividido en dos partes. Por un lado, la partición en teselas triangulares de la región a modelizar y, por otro, el modelo de superfície que se construirá sobre las teselas, o más en general, sobre regiones poligonales arbitrarias apuntadas por las teselas. Este artículo trata de la partición en triángulos y los problemas de diseño que plantea el manejo del gran volumen de datos que se espera (del orden de 10 Gbytes). El acceso para lectura de bloques de triángulos adyacentes debe ser rápido y la actualización debe ser fácil, incluyendo inserción y borrado de puntos y de segmentos requeridos. La triangulación escogida es la restringida de Delaunay. La figura 1 muestra un ejemplo de triangulación para un conjunto de datos muy conocido. Los vértices de los triángulos y los segmentos requeridos deben ser escogidos con criterios geomorfológicos, de modo que el modelo se adapte a la superfície del terreno con una precisión preestablecida. Ni estos vértices ni los segmentos requeridos deben estar condicionados por las estructuras de datos utilizadas para su almacenamiento.

1.2 Estructura DCEL

La DCEL es una estructura clásica [PF85] de la geometría computacional que permite almacenar cualquier grafo plano, en particular, un TIN. Es una lista de segmentos orientados. Para cada segmento se guardan apuntadores a sus dos vértices **V1** y **V2**, apuntadores al primer segmento vecino que encontramos girando alrededor de cada uno de los vértices en sentido antihorario **E1** y **E2**, opcionalmente puede contener un apuntador a cada una de las caras que comparten el segmento **F1** y **F2**. Como ha de almacenar una triangulación restringida, un atributo adicional indica si el segmento es requerido o no, **req** (figura 2).

1.3 Acceso a los triángulos

Para localizar el triángulo al que pertenece un punto podemos emplear el algoritmo de Lawson [CL77]: se parte de una posición actual (segmento actual); si el punto que buscamos se halla a su derecha seguiremos el apuntador **E1** y si se halla a la izquierda el apuntador **E2**. Se detecta que se ha alcanzado el triángulo buscado cuando se ha dado una vuelta completa alrededor de un mismo triángulo. Los apuntadores a los segmentos adyacentes permiten navegar por la estuctura viajando de un segmento al siguiente en tiempo constante $O(1)$, pero para acceder a un segmento lejano, en el caso de que la distribución de los vértices sea aproximadamente uniforme, el coste esperado es $O(n^{\frac{1}{2}})$ [GL85], donde n es el número de vértices del TIN, aunque existen disposiciones de puntos para las cuales el coste es $\Theta(n^2)$. Para distribuciones de puntos aproximadamente uniformes, la estructura de datos DCEL y la *quad-edge* de Guibas & Stolfi

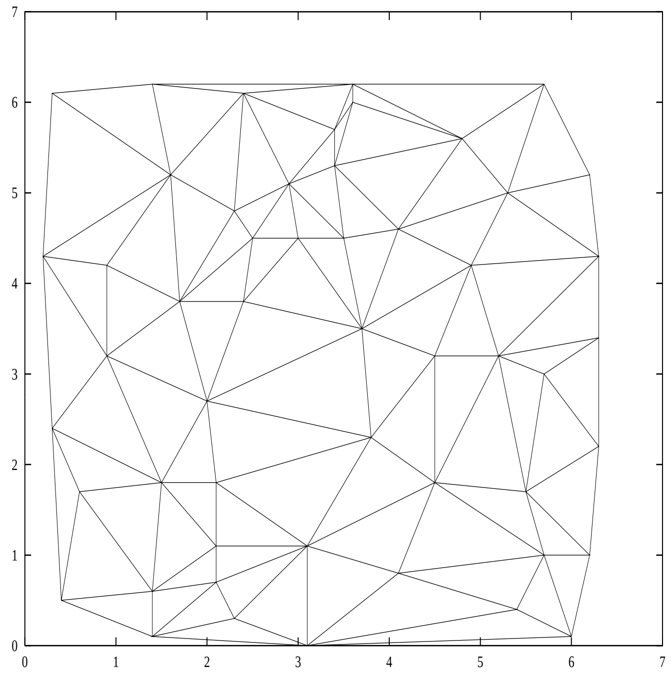

Figura 1: Triangulación de los datos de [Dav73]

simplificada para una superficie orientable [GL85] son equivalentes. Las rotaciones hacia la izquierda se pueden sustituir por un número adecuado de giros a la derecha. El ahorro de memoria compensa el mayor coste en tiempo.

El algoritmo de Lawson es adecuado para búsquedas locales, cuando todos los segmentos que hay que recorrer se hallan en la memoria central. Para grandes volúmenes de datos es necesario disponer de un método alternativo para acceder a triángulos no necesariamente contíguos ni "cercanos". Podemos ordenar los segmentos o superponer una estructura que permita acceder a los segmentos "lejanos" sin tener que seguir un camino de apuntadores intermedios que, en general, residirán en diferentes sectores de disco.

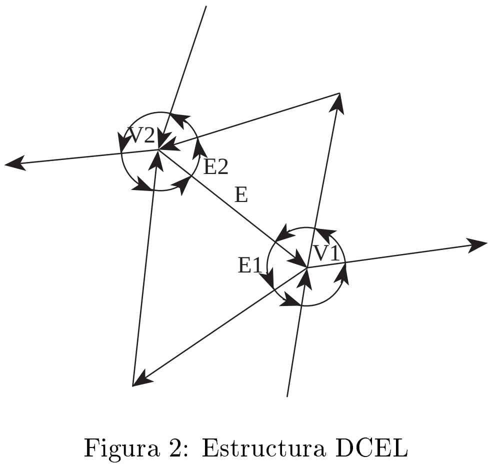

Figura 2: Estructura DCEL

2 Antecedentes: estructuras preexistentes relacionadas

2.1 Arbol de Delaunay

Se dice que una estructura TIN es dinámica cuando admite que sus vértices varíen con el tiempo, mediante inserción o borrado.

Boissonat y Teillaud [JM86] presentan una estructura semidinámica denominada árbol de Delaunay o estructura K [Sam90] que permite la inserción pero no el borrado de vértices. La estructura guarda la historia de la construcción incremental de la triangulación. Cuando se inserta un nuevo punto, los triángulos $\{T_i \mid i = 1, \cdots, n\}$ en conflicto con el nuevo punto M no se borran. Sea $R(M)$ la región que se formaría si uniésemos esos triángulos en conflicto y $F(M)$ su frontera. Un triángulo en conflicto T_i tendrá un lado F perteneciente a $F(M)$. Sea N el triángulo que comparte este lado F con T_i. Al insertar M se forman los nuevos triángulos uniendo los vértices de $F(M)$ con M. Sea S el triángulo recién creado con lado F. Para construir el árbol de Delaunay se crean unos apuntadores desde T_i y N a S. S se denomina hijo de T_i e hijastro de N.

Cada triángulo puede tener un hijo y una lista de hijastros para cada una de sus aristas. Los triángulos pertenecientes a la triangulación definitiva son aquellos que no tienen hijos (aunque posiblemente tengan hijastros).

Devillers et al. [DO90] presentan, como desarrollo posterior a [JM86], un algoritmo de borrado que deja el árbol de Delaunay en el mismo estado en que se hallaría si el punto eliminado nunca hubiera existido. El árbol permite la inserción estocástica de puntos en tiempo $O(n \log n)$ y el borrado en $O(n \log \log n)$. El espacio ocupado por el árbol es $O(n)$ pero la constante de proporcionalidad es bastante grande. Por cada punto que se añade se destruyen j y se crean $j+2$ triángulos nuevos si el triángulo es interior y $j+1$ en el caso de que pertenezca a la envolvente convexa. El número de nodos en el árbol de Delaunay es algo menor del triple del número de triángulos de la triangulación de Delaunay resultante para inserciones aleatorizadas [JM86].

2.2 PMR quadtree

Nelson y Samet [NR86] presentan el *PMR (Polygonal Map Random) quadtree*. Esta estructura permite almacenar segmentos de manera exacta, sin fragmentarlos en las fronteras de las hojas. La no fragmentación de los segmentos es muy importante porque las limitaciones en la representación de los números reales provocan que los fragmentos de un mismo segmento no estén alineados.

Este *quadtree* es un *quadtree* de regiones en el cual cada nodo se subdivide en cuatro hojas de igual tamaño (por bipartición del recorrido en cada dimensión). Un segmento se asigna a todas las hojas por las que pasa. El nombre de random deriva de que el criterio de subdivisión es probabilístico: si el número de segmentos de un bloque supera un valor umbral cuando el segmento es añadido, se divide el bloque en cuadrantes una única vez. Como el número de segmentos incidentes sobre un mismo vértice no está acotado, en el caso de que este número alcanzase el umbral de subdivisión de un hoja, no podríamos mediante subdivisión reducir el número de segmentos que una hoja contiene. En consecuencia, la ocupación de algunas hojas puede ser mayor que el valor umbral de subdivisión y el quadtree resultante depende del orden en que se realiza la inserción.

Una estructura adecuada para almacenar los segmentos es la de listas encadenadas. La ocupación (número de puntos por nodo) promedio de las hojas de *quadtree* es baja, Nelson y Samet [Sam90] obtienen resultados de un medio del valor umbral de subdivisión (ver tabla 1). Este cálculo es complejo y los autores hacen varias hipótesis simplificadoras para llevarlo a cabo. Entre ellas, la hipótesis de que la proporción de nodos con una ocupación determinada es independiente del tamaño del nodo. Las discrepancias entre las estimaciones teóricas y los resultados experimentales aumentan a medida que aumenta el umbral de subdivisión.

Estudiando el comportamiento del *PR quadtree*, Nelson y Samet encontraron unos efectos que denominaron envejecimiento (*aging*) y oscilación en fase (*phasing*) [Sam90]. El envejecimiento hace referencia a que los nodos más antiguos son también mayores y tienen tendencia a estar más ocupados que los nodos más pequeños. El *phasing* se produce cuando se insertan puntos pertenecientes a una distribución uniforme. Los nodos tienden a llenarse y subdividirse con un periodo logarítmico que se repite cada

Umbral de subdivisión	Ocupación experimental	teórica
1	1.22	1.22
2	1.62	1.22
3	2.14	2.20
4	2.71	2.81
5	3.31	3.46
6	3.90	4.14
7	4.53	4.81
8	5.10	5.51

Tabla 1: Ocupación media del *PMR-quadtree*, según Samet 1990

vez que el número de puntos se multiplica por 4. Los nodos del mismo tamaño tienden a llenarse y subdividirse a la vez y esto provoca oscilaciones en la ocupación promedio con respecto al número de puntos.

3 Estructura DCEL modificada y quadtree que la apunta

En esta comunicación presentamos una estructura DCEL modificada que se almacena en un *quadtree* de regiones (*R-quadtree*) en el cual la regla de inserción es la siguiente:

Cada segmento se almacena en aquellas hojas de *quadtree* que contienen a alguno de sus vértices. Un mismo segmento puede almacenarse en una o dos hojas de *quadtree*, no necesariamente adyacentes. El criterio de hoja indica si la hoja se debe subdividir o no. Como en el caso del *PMR quadtree*, es un criterio de desbordamiento. Una hoja se subdivide en cuatro cuadrantes iguales cuando el número de vértices (internos) que contiene supera un cierto umbral (figura 3). La subdivisión según el número de vértices permite que este número nunca supere el umbral. Como el número de segmentos incidentes sobre estos vértices es desconocido, éstos deben almacenarse en una lista de longitud variable.

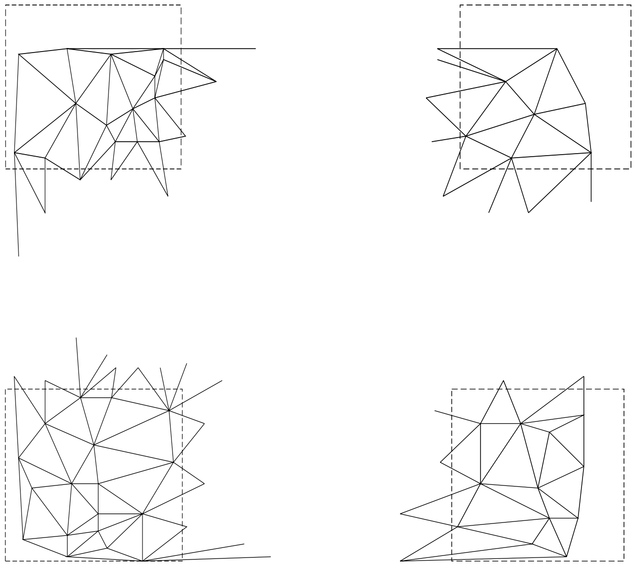

Figura 3: Partición de la triangulación de la figura 1 en cuatro hojas

Denominaremos segmentos transbloque a aquellos segmentos cuyos vértices pertenecen a dos hojas diferentes. Estos segmentos aparecen duplicados en las dos hojas. Diremos que un segmento transbloque tiene un segmento homólogo en otra hoja y sus atributos estarán distribuidos entre sus dos homólogos. Llamaremos vértice foráneo al vértice de un segmento transbloque que por sus coordenadas pertenece a la otra hoja. El número de vértices foráneos tampoco está acotado. Por razones de eficiencia una hoja de *quadtree* almacenará también la lista de vértices foráneos en otra lista de longitud variable. Dado que en cada hoja almacenamos los dos vértices del segmento, podemos deducir por las coordenadas del vértice foráneo en qué hoja de quadtree se halla el segmento homólogo. La estructura de datos para los segmentos transbloque es algo diferente de la estructura del resto de los segmentos. El apuntador al segmento siguiente girando alrededor del vértice foráneo es sustituido por el índice de su segmento homólogo dentro de la lista de segmentos de la otra hoja de *quadtree*.

La estructura DCEL modificada es la siguiente:

V1 primer vértice

V2 segundo vértice

E1 segmento siguiente girando alrededor de V2 en sentido antihorario
o índice del segmento homólogo si X1 = true

E2 segmento siguiente girando alrededor de V2 en sentido antihorario
o índice del segmento homólogo si X2 = true

F1 cara situada a la izquierda

F2 cara situada a la derecha

req ¿segmento requerido?

X1 ¿es V1 vértice foráneo?

X2 ¿es V2 vértice foráneo?

La existencia de los apuntadores a segmento homólogo y la posibilidad de calcular la hoja a la que este segmento pertenece a través de las coordenadas del vértice foráneo permiten encontrar la dirección del segmento homólogo en tiempo $O(p)$ siendo p la profundidad del *quadtree*. La profundidad p máxima depende de la distancia mínima d entre dos conjuntos diferentes de c puntos, siendo c la capacidad de las hojas. Supongamos un *quadtree* con n puntos que cubre una región cuadrada de lado l, y tal que d es la distancia mínima entre dos puntos. La profundidad p estará dentro del siguiente intervalo:

$$\left\lceil \log_4 \frac{n}{c} \right\rceil \leq p \leq \log_4 \left[\frac{2}{\sqrt{3}} \frac{l^2}{cd^2} + \left(1 + \frac{1}{\sqrt{3}}\right) \frac{5}{2} \frac{l}{dc} + \frac{13}{4c} \right] \tag{1}$$

Para $l \gg d$ sólo el primer sumando de la cota superior es importante. La distancia mínima d es un valor mínimo preestablecido a partir del cual se considera que dos puntos son iguales. Si esta distancia mínima puede ser muy pequeña en comparación con l, conviene alterar el criterio de hoja para limitar la profundidad máxima p. Las hojas dejarán de subdividirse cuando alcancen una profundidad máxima preestablecida. En nuestra aplicación, la capacidad c es grande, los vértices tienen una distribución aproximadamente uniforme y la distancia promedio entre puntos vecinos es mucho mayor que d. En este caso, este límite se alcanzará con una probabilidad despreciable.

Se ha realizado un experimento numérico para ver cómo varía la profundidad máxima y la ocupación media de las hojas con el número de vértices para distintas capacidades de hoja. Se ha tomado un conjunto de datos reales obtenidos por restitución fotogramétrica y se han insertado en el *quadtree* en el mismo orden en que han sido capturados. La figura 4 muestra la ocupación media de las hojas, en número de puntos/capacidad. La figura 5 muestra la profundidad máxima alcanzada por el árbol. Los resultados para la ocupación son similares a los de la tabla 1. No hay un cambio cualitativo de comportamiento al variar la capacidad de la hoja. Tampoco observamos efectos de envejecimiento o de subdivisión en fase debido a que la inserción no es aleatoria. Los datos reales están ordenados según el orden de su captura y tardan más en estabilizarse que los puntos generados aleatoriamente.

Podemos clasificar este árbol como un *bucket point region quadtree* [Sam90] con información adicional: una lista de segmentos y una lista de vértices foráneos.

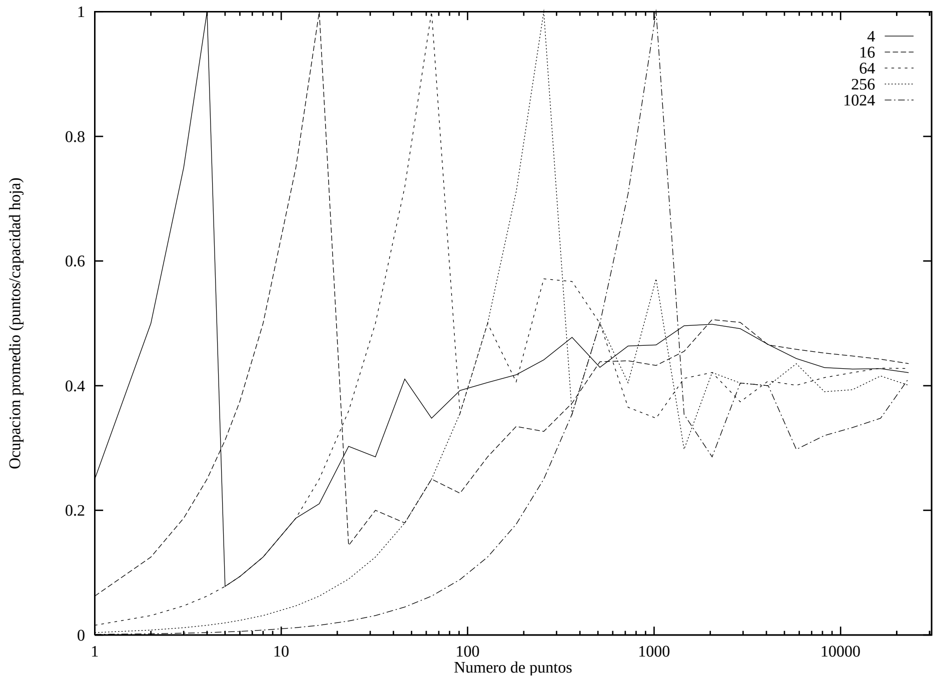

Figura 4: Ocupación promedio versus número puntos en el *quadtree* para diferentes capacidades de hoja: 4, 16, 64, 256 y 1024 puntos

4 Actualización de la triangulación

Es posible la actualización incremental (inserción y borrado) de puntos y segmentos requeridos. Los algoritmos serán una combinación con pequeñas modificaciones de los algoritmos que operan sobre un *bucket PR-quadtree* [Sam90] y de los algoritmos para la actualización incremental de la triangulación restringida de Delaunay almacenada en una estructura DCEL [dFL92].

La estructura *quadtree* sólo se ve afectada por la inserción y el borrado de vértices. Para insertar un vértice se busca a qué hoja pertenece siguiendo los apuntadores entre nodos del árbol desde la raíz. Si hay espacio en esa hoja, el vértice se añade a la lista correspondiente. Si no hay espacio, se subdivide la hoja en cuatro (con recolocación de los vértices internos, segmentos y vértices foráneos que contiene). A continuación se inserta el nuevo vértice. Para borrar un vértice hay que buscar a qué hoja pertenece, y, una vez localizada, buscar en la lista de vértices y eliminarlo. A continuación de un borrado suele hacerse una nueva inserción. Por esta razón no conviene volver a agrupar 4 hojas hermanas (operación refundir o *colapse*) aunque la suma de sus vértices sea menor que el umbral de subdivisión de un nodo. Sólo refundiremos cuando las 4 hojas hermanas estén vacías.

Ni la operación inserción ni el borrado de segmentos requeridos afectan a la estructura de árbol. Sólo implican a la estructura DCEL y supondrán modificaciones de las listas de segmentos y de vértices foráneos.

En todas las operaciones de actualización de la triangulación, ésta pasa por estados intermedios en los cuales algunos triángulos se han borrado y hay regiones poligonales cerradas sin triangular. Estos estados intermedios pueden ser representados tanto por la estructura DCEL como por la estructura presentada en este artículo.

Los algoritmos de actualización incremental de la triangulación operan aprovechando los atributos de los segmentos DCEL (tabla I). La estructura que presentamos modifica los atributos **E1** o **E2** en el caso de que los vértices **V1** o **V2** del segmento sean foráneos, y en su lugar se guarda el índice del segmento homólogo. Es necesario alterar los algoritmos de de Floriani & Puppo [dFL92] que operan sobre una DCEL para tener en cuenta esta modificación. Los atributos **E1** y **E2** son apuntadores que se utilizan

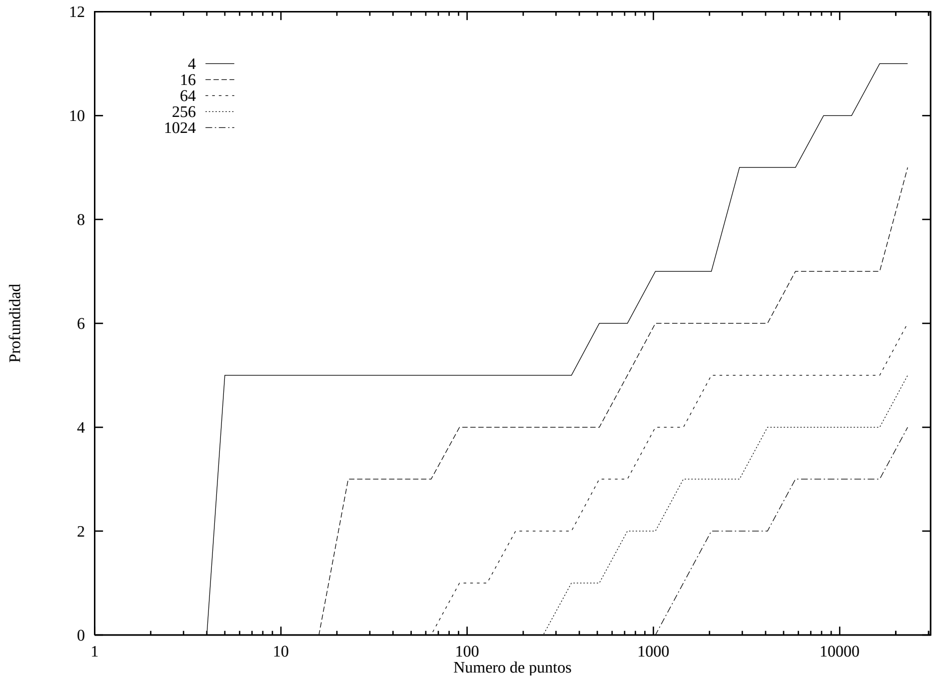

Figura 5: Profundidad máxima del *quadtree* versus número de puntos para diferentes capacidades de hoja: 4, 16, 64, 256 y 1024 puntos

para obtener el segmento siguiente girando desde el segmento actual **E** en sentido antihorario alrededor de los vértices **V1** o **V2** respectivamente. Denominaremos $Rot_1(E)$ y $Rot_2(E)$ a estas operaciones. Para la DCEL estas operaciones son inmediatas: $Rot_1(E) := E1(E)$ y $Rot_2(E) := E2(E)$. Con la modificación introducida $Rot_1(E)$ queda como sigue:

```
si V1 = foraneo entonces
   Devuelve(E1(E1(E)))
sino
   Devuelve(E1(E))
finsi
```

Figura 6: Operación $Rot_1(E)$

Se ha aprovechado la propiedad de que ambas representaciones de un segmento transbloque mantienen la misma orientación. Para $Rot_2(E)$ el algoritmo es análogo.

5 Discusión

Los árboles son estructuras muy adecuadas para almacenar grandes volúmenes de datos. El árbol de Delaunay ocupa mucho espacio y no ordena los datos ni siquiera parcialmente. Si el árbol se ha de guardar en memoria periférica, la lectura de datos de un árbol de Delaunay siguiendo la cadena de apuntadores provocará muchas faltas de paginación. Por otra parte, la inserción no aleatoria puede crear un árbol desequilibrado muy ineficiente aún en el caso de que los vértices tengan una distribución uniforme. Esto es debido a que la partición del espacio depende tando de los datos como del orden en que se insertan. El espacio ocupado por toda la historia de la triangulación es, como ya se ha indicado, muy grande.

El *PMR-quadtree* no mantiene la DCEL y tiene una gran redundancia al repetir cada segmento en todas las hojas que corta.

La estructura propuesta resuelve bien la ordenación parcial de los datos. Si la capacidad de la lista de vértices es suficientemente grande, el árbol no será muy profundo y podrá ser almacenado completamente en memoria. En este caso la recuperación de los datos correspondientes a un área rectangular pequeña podrá resolverse con un único acceso a disco. En estas condiciones el *quadtree* se comporta como un directorio que proporciona la dirección de disco en donde se hallan los datos. Como contrapartida, al aumentar la capacidad c de la lista de vértices, se degrada el tiempo requerido para la inserción incremental de puntos en la triangulación. Esto se debe a que la redistribución de datos que se produce cuando se subdivide una hoja es un proceso secuencial con un coste $O(c)$.

Los segmentos sólo se almacenan en las hojas a las que pertenecen sus vértices, aunque éstas no sean adyacentes. En el *PMR-quadtree* es necesario repetir el segmento en todas las hojas que corta debido a que es el propio árbol el encargado de almacenar las relaciones topológicas entre segmentos.

En la estructura de datos presentada las relaciones entre segmentos las mantiene la DCEL. El *quadtree* proporciona una partición adaptativa de la DCEL en bloques aproximadamente rectangulares. Además de permitir una aproximación rápida al bloque al que pertenece el triángulo buscado, el *quadtree* ofrece una ordenación parcial de los datos que permite la paginación de la triangulación entre la memoria central y la periférica. La partición es adaptativa en el sentido de que el número de bloques y su tamaño dependen de la densidad de vértices del área correspondiente.

6 Futuros desarrollos

En la actualidad estos algoritmos están siendo programados en Eiffel [B88]. La base de datos en la que se almacenará la información será O_2 [Tec95].

Con el fin de poder realizar actualizaciones masivas de datos y para mantener la compatibilidad con otros programas comerciales de modelado del terreno, hemos diseñado también algoritmos para la sustitución simultánea de bloques de datos interiores a un polígono.

Como ya se ha apuntado en la introducción, esta estructura es la base de un ambicioso sistema de modelado de la superfície topográfica que denominamos BasAlt [A.95]. La partición de la superficie en triángulos es una partición genérica. A cada tesela se le puede asociar el modelo de superficie que se desee: triángulo plano, NURB, superficie bilineal, etc. El modelo de superficie por defecto será el de triángulo plano. La utilización de la tecnología de orientación a objetos (sistemas Eiffel y O_2) permite ir añadiendo funcionalidades y refinar el modelo progresivamente.

References

[A.95] Ruiz A. Basalt: Base de dades altimètriques. memòria del projecte. Technical Report ?, Institut Cartogràfic de Catalunya., Barcelona, 1995.

[B88] Meyer B. *Object-oriented Software Construction.* Prentice Hall, New York, 1988.

[CL77] Lawson CL. Software for c^1 surface interpolation. In *Mathematical Software III.*, page ?, New York, 1977. Academic Press.

[dFL92] Puppo E de Floriani L. An on-line algorithm for constrained delaunay triangulation. *CVGIP: Graphical Models and Image Processing*, 54:290–300, 1992.

[DO90] Teillaud M Devillers O, Meiser S. Fully dynamic delaunay triangulation in logarithmic expected time per operation. Technical Report 1349, INRIA, Rocquencourt, France, 1990.

[GL85] Stolfi J Guibas LJ. Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams. *ACM Transactions on Graphics*, 4(2):74–123, 1985.

[JM86] Boissonat J.D. and Teillaud M. A hierarchical representation of objects: The delaunay tree. In *Second ACM Symposium on Computational Geometry in Yorktown Heights*, page ?, 1986.

[NR86] Samet H Nelson RC. A consistent hierarchical representation for vector data. In *Proc. SIGGRAPH'86 Conference.*, pages 197–206, 1986.

[PF85] Shamos MI Preparata FP. *Computational Geometry: An Introduction.* Springer, New York., 1985.

[Sam90] Hanan Samet. *The Design and Analysis of Spatial Data Structures.* Addison Wesley., Reading, MA, USA, 1990.

[Tec95] O_2 Technology. *The O_2 user manual.* O_2 Technology, Versailles. France., 1995.