Maintaining Partitions of Dynamic Planar Meshes *

L. G. Aleksandrov

Center of Informatics, Bulgarian Academy of Sciences, Sofia, Bulgaria

H. N. Djidjev

Department of Computer Science, Rice University, Houston, U.S.A.

Introduction. The solution of partial differential equations by the finite elements method on a parallel computer with p processors requires an efficient strategy to partition an unstructured mesh into p submeshes. The goal is to balance the computational load on the processors and to minimize the communication cost. Good mesh partitioning should satisfy the following properties: (i) each region of the partition should have approximately n/p elements, where n is the number of elements in the original mesh, and (ii) the number of vertices on boundaries between regions should be small.

In many time-dependent applications, after the initial partition, the data might need to be modified and then reallocated to the processors. Thus the problem for dynamic load balancing arises. For instance, in solving partial differential equations, the mesh might need to be modified (refined or coarsened) after every few time steps. Refinement types of modifications include adding a new point or an edge inside an existing element, adding a new point onto an existing edge, edge or face flipping, and others (Figure 1).

No deterministic algorithm is currently known that can efficiently recompute a new partition after modification of the mesh. The algorithm of Armon and Reif [2] is randomized and works only for plane graphs whose embedding meets certain criteria.

The main difficulty for designing a fast deterministic algorithm for maintaining partitions of dynamic planar graphs is related to the fact that most known algorithms use a breadth-first search as a basic step of the computation. There are no known algorithms that can dynamically maintain a breadth-first tree of a planar graph in sublinear time.

In this paper we develop a deterministic algorithm for maintaining partitions that avoids recomputation of breadth-first trees and works in $O(\log n)$ time.

*This work was partially supported by NSF grant No. CCR-9409191 and EPA grant R82-5207-01-0.

Main result. It will be more convenient to describe our problem in the terminology of plane graphs. We assume that the original graph has n faces whose size (number of edges on the boundary) is bounded by some constant d and that no update operation produces a face of size larger than d. An *ε-separator of G* we call a set of vertices whose removal leaves no connected component of more than εn faces. We consider update operation of the following types: (a) *insert_vertex*: inserting a new vertex v into an existing edge; (b) *delete_vertex:* replacing a sequence (u, v) and (v, w) of edges where the degree of v is two by a new edge (u, w) and removing v; (c) *insert_edge:* adding a new edge joining a pair of non-adjacent vertices belonging to the same face; and (d) *delete_edge:* deleting an edge whose endpoints have degrees greater than two. Note that the update operations are powerful enough so that the original graph G can be transformed into any other connected plane graph with no vertices of degree 1 by a sequence of these operations.

We have proved the following theorem.

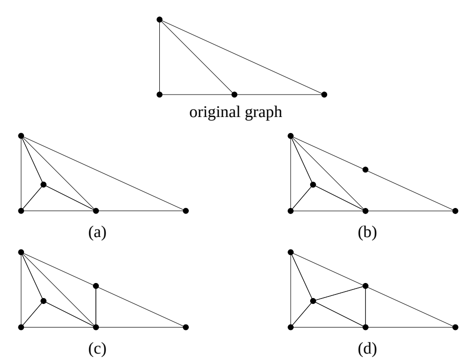

Figure 1: Mesh refinement operations: (a) adding a point inside a face; (b) adding a point onto an edge; (c) adding an edge; (d) edge flipping.

Theorem 1 *Let G be a plane graph with n faces whose size does not exceed a constant d and $\delta > 0$. Then a data structure exists that (i) can be constructed in $O(n)$ time, (ii) supports operations* insert_vertex, delete_vertex, insert_edge *and* delete_edge *in $O(\log n)$ time assuming the maximum face size of the resulting graph is bounded by d, and (iii) given any $\varepsilon > 0$ determines in $O(\log n)$ time an ε-separator of the current graph that can be extracted in $O(|S|)$ additional time, where $|S| = O(\sqrt{n^{1+\delta}/\varepsilon})$ is the size of the separator.*

Note that by [1] the bound on $|S|$ is within $O(n^{\delta/2})$ factor from the best possible.

Sketch of algorithm. Our approach is based on a representation of the partition of the current version of G as a hierarchy of nested partitions, where the partition on the lowest level has regions corresponding to the faces of G and the partition on the highest level has $O(1)$ regions. Moreover, each non-leaf node corresponds to a subgraphs of G. By updating the partition at an appropriately high level, we can reduce the update time by taking advantage of the smaller size of the graph at that level.

The preprocessing algorithm constructs the hierarchy of partitions using the following result. We call a partition a *regular h-partition* if each region (i) is either connected (i.e., consists of a single connected component), or has at most two neighboring regions, (ii) has at most h faces, and (iii) has at most $\sqrt{h}$ neighboring regions.

Lemma 1 *Let G be a plane graph with n faces whose maximal size is d. For any integer $h \geq d^2$ and $\varepsilon > 0$, a regular h-partition $\mathcal{R}$ such that $|\mathcal{R}| \leq cn/h$ can be constructed in $O(n)$ time.*

The preprocessing algorithm works as follows. Apply Lemma 1 with parameter h on the original graph. Remove the non-boundary vertices and compress the boundary of each region R of the resulting partition so that there is no vertex of degree two. (Regions consisting of more than one connected components require additional processing.) This transforms R into a single face denoted by $f(R)$. Also, for each edge e on $f(R)$ there is a pointer to the path on the boundary of R that has been contracted to e. According to Lemma 1 and our construction, the resulting plane graph $G^{(1)}$ has no more than cn/h faces and the size of each face does not exceed $\sqrt{h}$. Thus, Lemma 1 can be applied on $G^{(1)}$ with the same parameter h, producing a regular h-partition of $G^{(1)}$. We repeat the above steps until a graph $G^{(l)}$ with no more than h faces is obtained.

The resulting hierarchy of partitions has no more than $\log_h n + 1$ levels and the time for its construction is $O(n)$. The hierarchy is stored as a tree T, where each node v of T corresponds to a face of a partition and the parent of v is the node representing a face $f(R)$ such that the face corresponding to v is in R.

We have shown that the nodes at level k describe an ε_k-separator of the current graph G, where $\varepsilon_k = h^k/n$ and n is the number of faces in G. The separator itself can be extracted in time proportional to its size by repeatedly replacing edges of the separator by the corresponding paths on a lower level, starting with the set of all edges associated with nodes on level k. The size of the separator is $O(\sqrt{\frac{n^{1+\log c/\log h}}{\varepsilon_k}})$, where c is the constant of Lemma 1. Choosing $h = c^{1/\delta}$, we get the bound claimed in Theorem 1.

For maintaining T after an update operation, we search the tree to find a node u at the lowest possible level, say j, whose balance is violated as a result of the operation. We show that all unbalanced nodes will belong to a simple path between a leaf and the root of T. A node is defined to be *unbalanced* if, informally, it has too many children, or the face associated with it has a large boundary or too many adjacent faces. To restore the balance at u, we consider the subgraph $G(u)$ at level $j-2$ consisting of all faces associated with nodes descendants of u. The algorithm from Lemma 1 is then applied twice on $G(u)$ (see the algorithm that initially constructs T), producing a tree of height two that replaces the corresponding subtree of T rooted at u. The same procedure is repeated with all ancestors of u in increasing order of levels. We show that this algorithm restores the balance of all nodes of T in $O(\log n)$ time.

References

[1] Lyudmil Aleksandrov and Hristo N. Djidjev. Linear algorithms for partitioning embedded graphs of bounded genus. *SIAM Journal on Discrete Mathematics*, 9:129–150, 1996.

[2] D. Armon and J. Reif. A dynamic separator algorithm. In *Proc. 3rd Worksh. Algorithms and Data Structures*, pages 107–118. Lecture Notes in Computer Science 709, Springer-Verlag, Berlin, 1993.