Data Structures for Simplicial Multi-Complexes

Leila De Floriani, Paola Magillo, and Enrico Puppo

Dipartimento di Informatica e Scienze dell'Informazione – Università di Genova
Via Dodecaneso, 35, 16146 Genova, ITALY
{deflo,magillo,puppo}@disi.unige.it

Abstract. The *Simplicial Multi-Complex* (SMC) is a general multiresolution model for representing k-dimensional spatial objects through simplicial complexes. An SMC integrates several alternative representations of an object and offers simple methods for handling representations at variable resolution efficiently, thus providing a basis for the development of applications that need to manage the level-of-detail of complex objects. In this paper, we present general query operations on such models, we describe and classify alternative data structures for encoding an SMC, and we discuss the cost and performance of such structures.

1 Introduction

Geometric cell complexes (meshes) have a well-established role as discrete models of continuous domains and spatial objects in a variety of application fields, including Geographic Information Systems (GISs), Computer Aided Design, virtual reality, scientific visualization, etc. In particular, simplicial complexes (e.g., triangle and tetrahedra meshes) offer advantageous features such as adaptivity to the shape of the entity, and ease of manipulation.

The accuracy of the representation achieved by a discrete geometric model is somehow related to its resolution, i.e., to the relative size and number of its cells. At the state-of-the-art, while the availability of data sets of larger and larger size allows building models at higher and higher resolution, the computing power and transmission bandwidth of networks are still insufficient to manage such models at their full resolution. The need to trade-off between accuracy of representation, and time and space constraints imposed by the applications has motivated a burst of research on *Level-of-Detail (LOD)*. The general idea behind LOD can be summarized as: *always use the best resolution you need – or you can afford – and never use more than that.* In order to apply this principle, a mechanism is necessary, which can "administrate" resolution, by adapting a mesh to the needs of an application, possibly varying its resolution over different areas of the entity represented.

A number of different LOD models have been proposed in the literature. Most of them have been developed for applications to terrain modeling in GISs (see, for instance, [1, 4, 9]) and to surface representation in computer graphics and virtual reality applications (see, for instance, [10, 8, 15, 7]), and they are strongly

characterized by the data structures and optimization techniques they adopt as well as custom tailored to perform specific operations, and to work on specific architectures. In this scenario, developers who would like to include LOD features in their applications are forced to implement their own models and mechanisms. On the other hand, a wide range of potential applications for LOD have been devised, which require a common basis of operations (see, e.g., [3]). Therefore, it seems desirable that the LOD technology is brought to a more mature stage, which allows developers to use it through a common interface, without the need to care about many details.

In our previous work, we have developed a general model, called a *Simplicial Multi-Complex (SMC)*, that can capture all LOD models based on simplicial complexes as special cases [13, 5, 14]. Based on such model, we have built systems for managing the level of detail in terrains [2], and in free-form surfaces [3], and we are currently developing an application in volume visualization.

In this paper, we consider general operations that can be performed on LOD models and propose an analysis of cost and performance of their encoding data structures. Trade-off between cost and performance is a key issue to make the LOD technology suitable to a wide spectrum of applications and architectures in order to achieve a more homogeneous and user-transparent use of LOD.

The Simplicial Multi-Complex is briefly described in Section 2, and general query techniques on such model are outlined in Section 3. In Section 4, we analyze the spatial relations among entities in the SMC, which are fundamental to support queries and traversal algorithms. In Section 5, we analyze different data structures to encode SMCs in the general case, as well as in special cases, and we discuss both the cost of such data structures, and their performance in supporting the extraction of spatial relations. In Section 6, we present some concluding remarks.

2 Simplicial Multi-Complexes

In this section, we briefly review the main concepts about the Simplicial Multi-Complex, a dimension-independent multiresolution simplicial model which extends the Multi-Triangulation presented in [13, 5, 14]. For the sake of brevity, this subject is treated informally here. For a formal treatment and details see [11].

In the remainder of the paper, we denote with k and d two integer numbers such that $0 < k \leq d$. A *k-dimensional simplex* σ is the locus of points that can be expressed as the convex combination of $k+1$ affinely independent points in $\mathbb{R}^d$, called the *vertices* of σ. Any simplex with vertices at a subset of the vertices of σ is called a *facet* of σ. A *(regular) k-dimensional simplicial complex* in $\mathbb{E}^d$ is a finite set Σ of k-simplices such that, for any pair of distinct simplices $\sigma_1, \sigma_2 \in \Sigma$, either σ_1 and σ_2 are disjoint, or their intersection is the set of facets shared by σ_1 and σ_2. In what follows, a k-simplex will be always called a *cell*, and we will deal only with complexes whose domain is a manifold (also called *subdivided manifolds*).

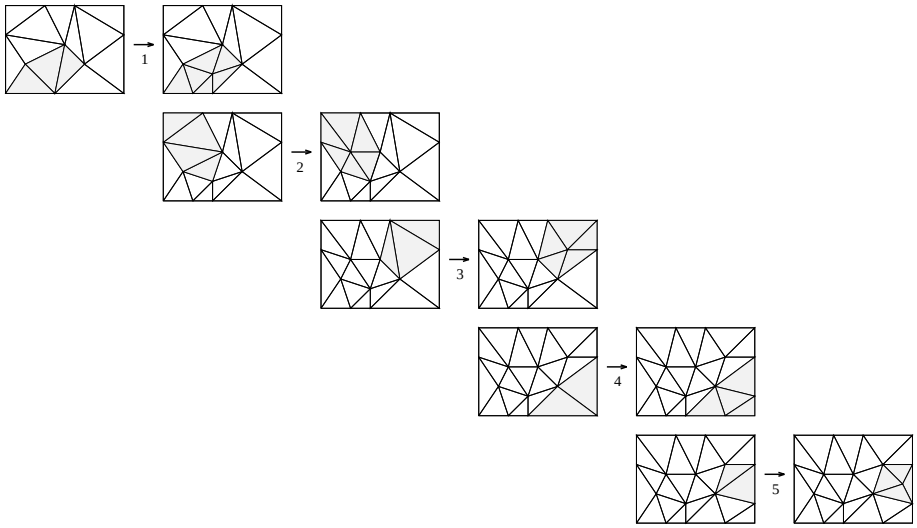

Fig. 1. A sequence of five updates (numbered 1...5) progressively refining an initial coarse triangle mesh. The area affected by each update is shaded.

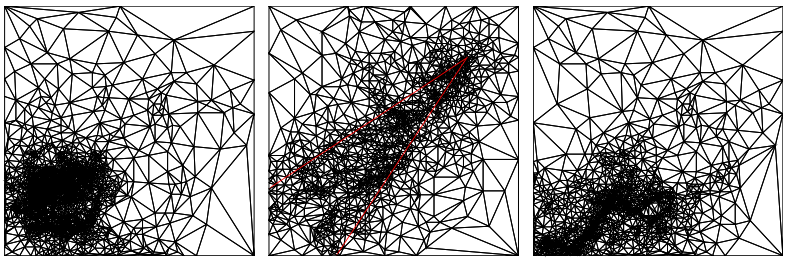

Fig. 2. Three meshes extracted from a two-dimensional SMC representing a terrain (top view). (a) The triangulation has the highest possible resolution inside a rectangular window, and the lowest possible resolution outside it. (b) Resolution inside a view frustum (wedge) is decreasing with the distance from its focus point, while it is arbitrarily low outside it. (c) Resolution is high only in the proximity of a polyline.

The intuitive idea behind a *Simplicial Multi-Complex* (SMC) is the following: consider a process that starts with a coarse simplicial complex and progressively refines it by performing a sequence of local updates (see Figure 1). Each *local update* replaces a group of cells with another group of cells at higher resolution. An update U_2 in the sequence *directly depends* on another update U_1 preceding it if U_2 removes some cells introduced with U_1. The *dependency relation* between updates is defined as the transitive closure of the direct dependency relation. Only updates that depend on each other need to be performed in the given order; mutually independent updates can be performed in arbitary order. For instance, in the example of Figure 1, updates 3 and 4 are mutually independent, while update 5 depends on both; thus, we must perform update 4 first, then followed by 3 and 5.

An SMC abstracts from the totally ordered sequence by encoding a *partial order* describing the *mutual dependencies* between pairs of updates. Updates forming any subset closed with respect to the partial order, when performed in a consistent sequence, generate a valid simplicial complex. Thus, it is possible to perform more updates in some areas, and fewer updates elsewhere, hence obtaining a complex whose resolution is variable in space. Such an operation is known as *selective refinement*, and it is at the basis of LOD management. A few results of selective refinement from an SMC representing a terrain are shown in Figure 2.

An SMC is described by a directed acyclic graph (DAG). Each update is a node of the DAG, while the arcs correspond to direct dependencies between updates. Each arc is labeled with the collection of all cells of its source node that are removed by its destination node. For convenience, we introduce two further nodes: a *root* corresponding to the update creating the initial coarse complex, which is connected with an arc to each update that removes some if its cells; and a *drain*, corresponding to the final deletion of the complex obtained by performing all updates, which is connected with an arc from each update that creates some of its cells. Also such arcs are labeled by cells in a consistent way. Figure 3 shows the SMC corresponding to the collection of updates described in Figure 1.

A *front* of an SMC is a set of arcs containing exactly one arc on each directed path from the root (see Figure 3). Since the DAG encodes a partial order, we say that a node is *before* a front if it can be reached from the root without traversing any arc of the front; otherwise the node is said to be *after* the front. Nodes lying before a front define a consistent set of updates, and the corresponding simplicial complex is formed by all cells labeling the arcs of the front [11]. By sweeping a front through the DAG, we obtain a wide range of complexes, each characterized by a different resolution, possibly variable in space.

In the applications, often an SMC is enriched with attribute information associated with its cells. Examples are approximation errors (measuring the distance of a cell from the object portion it approximates), colors, material properties, etc.

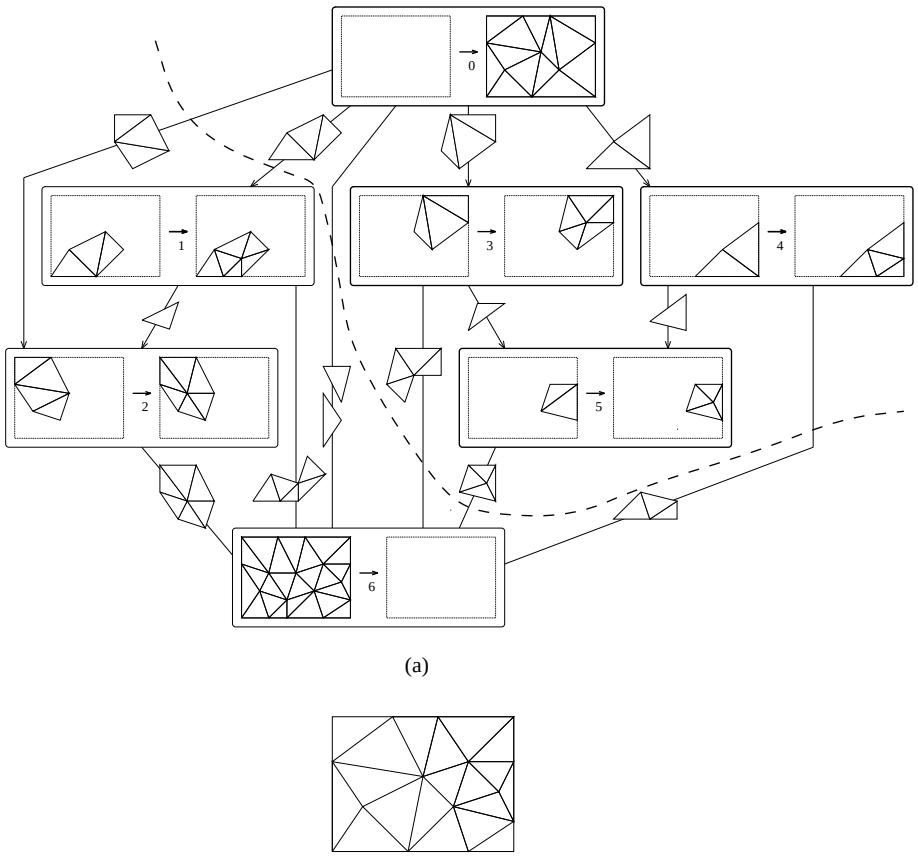

Fig. 3. (a) The SMC built over the partially ordered set of mesh updates of Figure 1. Each node represents an update, and it shows the two sets of simplices removed and created in the update. Each arc represents the dependency between two updates, and it is labelled by the triangles created in the first update, which are removed in the second update. A front on the SMC contains the arcs intersected by the thick dashed line; nodes lying before the front are highlighted. (b) The triangle mesh associated with the front.

3 A Fundamental Query on an SMC

Since an SMC provides several descriptions of a spatial object, a basic query operation consists of selecting a complex which represents the object according to some user-defined resolution requirements. This basic query provides a natural support to variable resolution in many operations, such as:

- *point location*, i.e., finding the cell that contains a given point and such that its resolution meets some user-defined requirements;
- *windowing*, i.e., finding a complex, that represents the portion of the object lying inside a box, at a user-defined resolution;
- *ray casting*, i.e., finding the cells that intersect a given ray at a user-defined resolution;
- *perspective rendering*: in this case, a complex is generated which represents the portion of the object lying inside the view frustum, and whose resolution is higher near the viewpoint and decreases with the distance from it;
- *cut*, i.e., sectioning with a hyperplane: the section is computed by first retrieving the cells that intersect the given hyperplane and have a specific resolution.

In the most general case, resolution requirements are expressed through a *resolution filter*, which is a user-defined function R that assigns to each cell σ of the SMC a real value $R(\sigma)$. Intuitively, a resolution filter measures the "signed difference" between the resolution of a cell and that required by the application: $R(\sigma) > 0$ means that the resolution of σ is not sufficient; $R(\sigma) < 0$ means that the resolution of σ is higher than necessary. A cell such that $R(\sigma) \leq 0$ is said *feasible*.

For example, the meshes depicted in Figure 2 satisfy the following resolution filters: in (a), R is negative for all cells outside the window, zero for all cells inside it that are at the highest resolution, and positive for all others; in (b), R is negative for all cells outside the view frustum, while for a cell σ inside it, R is decreasing with resolution of σ, and with its distance from the focus point; in (c), R is negative for all cells not intersecting the polyline, zero for all cells intersecting it that are at the highest resolution, and positive for all others.

The basic query on an SMC consists of retrieving the simplicial complex of *minimum size* (i.e., composed by the smallest number of cells) which satisfies a given resolution filter R (i.e., such that all its cells are feasible with respect to R). Variants of this query are also described in [11]. The basic query can be easily combined with a culling mechanism, which extracts only the subcomplex intersecting a given *Region Of Interest (ROI)*. This *localized* query permits to implement operations like point location, windowing, etc.

Algorithms for mesh extraction [13, 3, 14, 11] sweep a front through the DAG, until an associated complex formed by feasible cells is found. Minimum size is guaranteed by a front that lies as close as possible to the root of the SMC. In the case of a localized query, spatial culling based on a ROI is incorporated in the DAG traversal, hence using the structure of the SMC also as a sort of spatial index. The key operations used by extraction algorithms consist in either advancing the front after a node, when the resolution of the complex over that

area is not sufficient, or moving it before a node when the resolution over that area is higher than required.

The key issues that have impact on the performance of such algorithms are: the evaluation of the resolution function, which is application-dependent; and the evaluation of mutual relations that occur among different entities of the SMC. The cost of computing such relations is highly dependent on the amount of information stored in the data structure.

4 Relations in a Simplicial Multi-Complex

In some applications, e.g., in computer graphics, it is often sufficient to represent a simplicial complex by the collection of its cells, where each cell is described by its vertices and its attributes. In other applications, e.g., in GIS, in CAD, or in scientific visualization, topological relations among vertices and cells of the mesh must be maintained as well. A common choice is the *winged* data structure, which stores, for each cell, the $(k+1)$ cells adjacent to it along its $(k-1)$-facets [12]. Building the winged data structure for the mesh produced as the result of a query on the SMC can be more or less expensive, depending on the data structure used to encode the SMC.

In the following, we discuss the relations among the elements of an SMC, which are needed in the traversal algorithms, and in building the winged data structure for the output mesh.

There are essentially three kinds of relations in an SMC:

- *Relations on the DAG*: they define the structure of the DAG describing the SMC by relating its nodes and arcs.
- *Relations between the DAG and the cells of the SMC*: they define the connections between the elements of the DAG (arcs and nodes) and the cells forming the SMC; in the definition given in Section 2, such a connection is defined by labeling each arc of the DAG with the cells created by its source node that are removed by its destination node.
- *Relations between the simplices of the SMC*: they define the relations among vertices and cells in the SMC.

The relations on the DAG are the standard relations in a directed graph: *Node-Arc (NA)*, which associates with a node its incoming and its outgoing arcs; and *Arc-Node (AN)*, which associates with an arc its source and its destination.

The relations between the DAG and the cells of the SMC can be defined as follows:

- *Arc-Cell (AC)* relation, which associates with an arc of the DAG the collection of the cells labeling it.
- *Cell-Arc (CA)* relation, which associates with a cell σ of the SMC the arc of the DAG whose label contains σ.
- *Node-Cell (NC)* relation, which associates with a node U the cells created and deleted by the corresponding update.

– *Cell-Node (CN)* relation, which associates with a cell σ the node U introducing σ in its corresponding update, and the node U' removing σ.

The relations between the simplices in an SMC we are interested in are:

– the relation between a cell and its vertices, that we call *Cell-Vertex (CV)* relation;
– the *adjacency* relation between two cells, which share a $(k-1)$-facet, that we call a *Cell-Cell (CC)* relation.

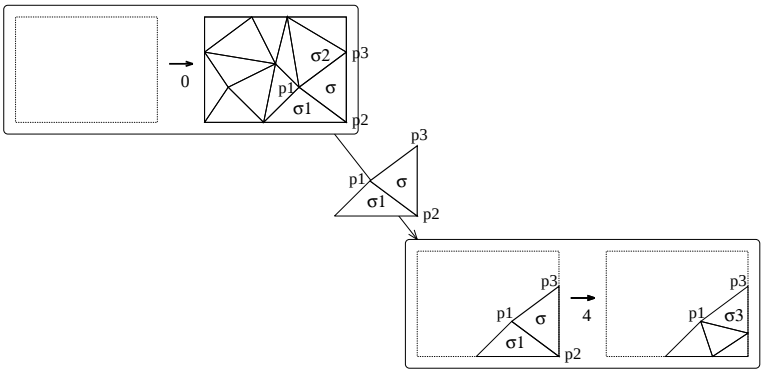

Fig. 4. A fragment of the SMC of Figure 3 and CC relations involving simplex σ. At edge p_1p_2, relation co-CC_1 and co-CC_2 both give simplex σ_1; relations counter-CC_1 and counter-CC_2 are not defined. At edge p_2p_3 no CC relation is defined. At edge p_3p_1, relation co-CC_1 is not defined, relation counter-CC_1 gives σ_3; relation co-CC_2 gives σ_2 and counter-CC_2 is not defined.

Since not all cells sharing a $(k-1)$-facet in the SMC can coexist in a cell complex extracted from it, we specialize the CC relation further into four different relations that will be used in the context of data structures and algorithms discussed in the following (see also Figure 4). Given two cells σ_1 and σ_2 that share a $(k-1)$-facet σ'':

1. σ_1 and σ_2 are *co-CC_1* at σ'' if and only if σ_1, σ_2 have been removed by the same update (i.e., they label either the same arc or two arcs entering the same node);
2. σ_1 and σ_2 are *co-CC_2* at σ'' if and only if σ_1, σ_2 have been created by the same update (i.e., they label either the same arc or two arcs leaving the same node);
3. σ_2 is *counter-$CC_{1,2}$* to σ_1 at σ'' if and only if, σ_2 is created by the update that removes σ_1 (i.e., the arc containing σ_1 and that containing σ_2 enter and leave the same node, respectively);

4. σ_2 is *counter-*$CC_{2,1}$ to σ_1 at σ'' if and only if σ_2 is removed by the update that creates σ_1 (i.e., the arc containing σ_1 and that containing σ_2 leave and enter the same node, respectively).

Relations co-CC_1 and counter-$CC_{1,2}$ are mutually exclusive: a k-simplex cannot have both a co-CC_1, and a counter-$CC_{1,2}$ cell at the same $(k-1)$-facet. The same property holds for relations co-CC_2 and counter-$CC_{2,1}$. The above four relations do not capture all possible CC relations among cells in an SMC, but they are sufficient to support efficient reconstruction algorithms, as explained in the following.

Relations CV and CC, defined in the context of a mesh extracted from an SMC by the algorithms described in Section 3, also characterize the winged data structure. Now, let us assume that we want to encode our output mesh through such a data structure. We have three options:

1. *Adjacency reconstruction as a post-processing step*: the extraction algorithm returns just a collection of cells and vertices, together with the CV relation; pairs of adjacent (CC) cells in the output mesh are found through a sorting process. This takes $O(m(k+1)\log(m(k+1)))$ time, where m is the number of cells in the mesh, and k is the dimension of the complex.
2. *Incremental adjacency update*: the pairs of adjacent cells in the output mesh are determined and updated while traversing the SMC, encoded with a data structure that maintains the four relations co-CC_1, co-CC_2, counter-$CC_{1,2}$ and counter-$CC_{2,1}$.
   In the extraction algorithms, when the current front is advanced after a node, the pairs of mutually adjacent new cells introduced in the current mesh are determined by looking at co-CC_2 relations in the SMC; the adjacency relations involving a new cell, and a cell that was already present in the output mesh, are determined by using relation counter-$CC_{2,1}$ and adjacency relations of the old cells replaced by the update. Symmetrically, when the current front is moved before a node, relations co-CC_1 and counter-$CC_{1,2}$ permit updating adjacency relations in the current mesh. The total time is linear in the number of cells swept from one side to the other of the front.
3. *Incremental adjacency reconstruction*: same as approach 2, but without encoding CC relations of the SMC.
   In this case, when sweeping a front through a node (either forward or backward), a process of adjacency reconstruction similar to that used in approach 1 is applied, locally to the part of the current complex formed by the new cells introduced in the current mesh, and the cells adjacent to those deleted by the update operation. The time required is $O(n_{sweep}\log M)$, where n_{sweep} is the number of swept cells, and M is the maximum number of cells removed and created by the update contained in a swept node.

5 Data Structures

Encoding an SMC introduces some overhead with respect to maintaining just the mesh at the highest possible resolution that can be extracted from it. This is

indeed the cost of the mechanism for handling multiresolution. However, we can trade-off between the space requirements of a data structure and the performance of the query algorithms that work on it.

From the discussion of previous sections, it follows that basic requirements for a data structure encoding an SMC are to support selective refinement (as outlined in Section 3), and to support the extraction of application-dependent attributes related to vertices and cells. Moreover, a data structure should support the efficient reconstruction of spatial relationships of an output mesh, for those applications that require it.

In the following subsections, we describe and compare some alternative data structures that have different costs and performances. Those described in Sections 5.1 and 5.2 can be used for any SMC, while those described in Section 5.3 can be used only for a class of SMCs built through specific update operations.

5.1 Explicit data structures

An *explicit* data structure directly represents the structure of the DAG describing an SMC. It is characterized by the following information:

- For each vertex, its coordinates.
- For each cell: the Cell-Vertex and the Cell-Arc relations, plus possibly a subset of *Cell-Cell* relations (as described below).
- For each node: the Node-Arc relation.
- For each arc: the Arc-Node relation, and the Arc-Cell relation.

Depending on the specific application, additional information may be attached to vertices and/or cells (e.g., approximation errors for simplices). Here, we do not take into account such extra information.

Assuming that any piece of information takes one unit, the space required by this data structure, except for adjacency relations and attributes, is equal to $d\mathtt{v} + (k+3)\mathtt{s} + 4\mathtt{a}$, where $\mathtt{v}$, $\mathtt{s}$ and $\mathtt{a}$ denote the number of vertices, cells, arcs in the SMC, respectively. Note that $4\mathtt{a}$ is the cost of storing the NA plus the AN relations (i.e., the DAG structure), while $2\mathtt{s}$ is the cost of storing the CA and AC relations, i.e. information connecting the DAG and the cells of the SMC.

We consider three variants of adjacency information that can be stored for each cell σ:

- *Full-adjacency*: all four adjacency relations are maintained: co-CC_1, co-CC_2, counter-$CC_{1,2}$ and counter-$CC_{2,1}$. For each $(k-1)$-facet of σ, co-CC_1 and counter-$CC_{1,2}$ are stored in the same physical link, since they cannot be both defined; similarily, co-CC_2 and counter-$CC_{2,1}$ are stored in the same physical link. Thus, we have $2(k+1)$ links for each simplex.
- *Half-adjacency*: only relations co-CC_2 and counter-$CC_{2,1}$ are stored, by using the same physical link for each edge e of σ, thus requiring $(k+1)$ links.
- *Zero-adjacency*: no adjacency relation is stored.

The version with full-adjacency can support incremental adjacency update (see approach 2 in Section 4). The version with half-adjacency can support incremental adjacency update only when advancing the front after a node. With zero-adjacency, adjacency reconstruction must be performed, either as a post-processing (approach 1), or incrementally (approach 3).

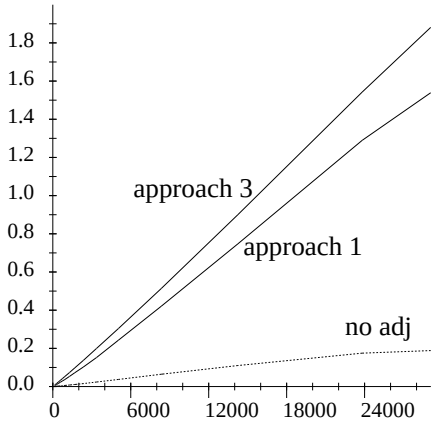

Fig. 5. Query times with adjacency reconstruction on a two-dimensional SMC using approach 1 and 3; the dotted curve represents query time without adjacency generation. The horizontal axis reports the number of triangles in the output mesh, the vertical axis execution times (in seconds).

Figure 5 compares the performance of query algorithms on the zero-adjacency data structure without adjacency generation, and with adjacencies reconstructed through approaches 1 and 3. Adjacency reconstruction increases query times of almost a factor of ten. Therefore, it seems desirable that adjacency information are maintained in the SMC data structure whenever they are necessary in the output mesh, provided that the additional storage cost can be sustained.

5.2 A Data Structure Based on Adjacency Relations

In this section, we describe a data structure that represents the partial order which defines the SMC implicitly, i.e., without encoding the DAG, but only using adjacency relations. The data structure stores just vertices and cells, and it maintains the following information:

- For each vertex: its coordinates.
- For each cell: the Cell-Vertex relation, and the four Cell-Cell relations, (using $2(k+1)$ links as explained in Section 5.1).

The space required is $d\mathtt{v} + 3(k+1)\mathtt{s}$.

Given a cell σ, removed by an update U, all other cells removed by U are found through co-CC_1 starting at σ. Among such cells, a cell σ'' is found which

has at least one counter-$CC_{2,1}$ simplex σ''. Finally, starting from σ'' (which is a cell created by U), all remaining cells created by U are found by using relation co-CC_2. These properties allow us to update the current mesh after any movement of the current front.

The size of the adjacency-based data structure is always larger than that of the explicit structure with zero-adjacency, while it is comparable with that of the explicit data structure encoding some adjacency. Note that the space needed to store adjacency relations tends to explode when the dimension k of the model increases. Implementing query algorithms on the adjacency-based structure is more involved than on the explicit structure, and it requires maintaining larger temporary structures for encoding the internal state (see [11] for details). Therefore, this data structure should be preferred over to the explicit ones only if adjacency relations are fundamental in the output structure, and the dimension of the problem is low. However, the performance of the query algorithms is likely to degrade with respect to the case of explicit data structures. Storage costs for $k = 2, 3$ are compared in Table 1.

k=d=2

structure	space
explicit (zero-adj)	$2\mathtt{v} + 5\mathtt{s} + 4\mathtt{a}$
explicit (half-adj)	$2\mathtt{v} + 8\mathtt{s} + 4\mathtt{a}$
explicit (full-adj)	$2\mathtt{v} + 11\mathtt{s} + 4\mathtt{a}$
adj-based	$2\mathtt{v} + 9\mathtt{s}$

k=d=3

structure	space
explicit (zero-adj)	$3\mathtt{v} + 6\mathtt{s} + 4\mathtt{a}$
explicit (half-adj)	$3\mathtt{v} + 10\mathtt{s} + 4\mathtt{a}$
explicit (full-adj)	$3\mathtt{v} + 14\mathtt{s} + 4\mathtt{a}$
adj-based	$3\mathtt{v} + 12\mathtt{s}$

Table 1. Space requirements of the explicit and the adjacency-based data structures for $k = 2, 3$.

5.3 Compressed Data Structures

Much of the cost of data structures presented in the previous sections is due to the explicit representation of the cells and of the cell-oriented relations in the SMC. Indeed, the total number of cells is usually quite larger than the number of vertices, arcs, and nodes involved in the model, and relations among cells and vertices are expensive to maintain: for instance, a cell needs $k + 1$ vertex references for representing relation CV.

In some cases, the structure of every update exhibits a specific pattern, which allows us to compress information by representing cells implicitly. Examples of update patterns commonly used in building LOD models for surfaces are: *vertex insertion*, which is performed by inserting a new vertex and retriangulating its surrounding polytope consequently; and *vertex split*, which is performed by expanding a vertex into an edge and warping its surrounding cells consequently.

Such update patterns are well defined in any dimension d, and they are depicted in Figure 6 for the two-dimensional case.

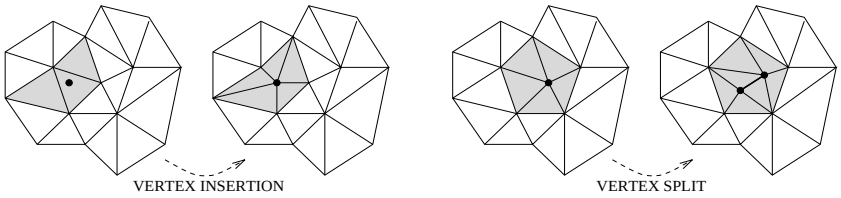

Fig. 6. Two types of update patterns that allow the design of compressed data structures for an MC. The shaded triangles are those involved in the update.

Since each update U exhibits a predefined pattern, the set of cells it introduces can be encoded by storing just a few parameters within U, that are sufficient to describe how the current complex must be modified when sweeping the front, either forward or backward, through U. The type and number of parameters depend on the specific type of update for which a certain compressed data structure is designed.

The generic scheme for a compressed data structure encodes just the vertices, nodes, and arcs of an SMC. For vertices, it stores the same information as the standard explicit structure; for nodes and arcs it encodes the following information:

- For each node U: the Node-Arc relation, plus an implicit description of the cells defining the update described by U, i.e., an implict encoding of the combination of Node-Cell and Cell-Vertex relations.
- For each arc: the Arc-Node relation.

The space required (except for the implicit encoding of the NC relation combined with the CV one) is equal to $d\mathtt{v} + 4\mathtt{a}$.

Note that, since cells do not exist as individual entities, attribute information for them cannot be encoded explicitly. This means that, while the exact geometry of cells can be obtained through suitable mechanisms associated with update parameters, their attributes can only be approximated through information associated with nodes. In other words, attributes on a node U summarize attributes of the whole group of cells associated with it. In this sense, such a structure is *lossy*, because it cannot discriminate between attributes of different cells in the context of the same node. Since the evaluation of the resolution filter may depend on cell attributes, because of the approximation, a given cell σ may result unfeasible even if it was feasible, or viceversa. This fact may cause the extraction of a mesh that is either over- or under-refined with respect to the input requirements.

Another subtle issue, which affects the performance of the extraction algorithms, is the lack of information on the update that must be applied to refine

the mesh at a given cell. This is due to the fact that cells are associated with nodes, rather than with arcs. When sweeping the current front after a node U, the state of the current mesh, and the update information stored in U, allow us to determine which cells are removed, and which cells are created by U. All new cells are tagged with U as their creator. Let σ be a cell introduced by U. If σ is not feasible, then we should advance the front after the node U' that removes σ. Unfortunately, the data structure does not provide information on which child of U removes σ. In order to avoid cumbersome geometric tests to find U', we adopt a conservative approach that advances the front after all children of U. However, such an approach may lead to over-refine the extracted mesh with respect to the output of the same query answered on a general data structure. Similar problems arise when sweeping the current front before a node. See [11] for further details.

In the following, we describe in more detail two specific data structures for the case of vertex insertion. The first data structure applies to SMCs in arbitrary dimension d, while the second structure is specific for two-dimensional SMCs. Similar data structures for the case of vertex split can also be obtained, by combining the ideas presented here with the mechanism described in [8] for a single update.

A Structure for Delaunay SMCs We present a compressed structure designed for d-dimensional SMCs in $\mathbb{R}^d$, such as the ones used to represent the domain of scalar fields (e.g., for $d = 2$, the domain of terrains, or of parametric surfaces), based on Delaunay simplicial complexes. A simplicial complex is called a *Delaunay simplicial complex* if the circumsphere of any of its cells does not contain vertices in its interior. In two dimensions, Delaunay triangulations are widely used in terrain modeling because of the regular shape of their triangles and since efficient algorithms are available to compute them.

In a Delaunay SMC, the initial simplicial complex is a Delaunay one, and every other node U represents the insertion of a new vertex in a Delaunay complex. Thus, every extracted complex is a Delaunay complex.

For a set of points in general positions (no $d+2$ points are co-spherical), the Delaunay complex is unique; thus, a Delaunay complex is completely determined by the set of its vertices, and the update due to the insertion of a new vertex is completely determined by the vertex being inserted. The data structure encodes cells in the following way:

- at the root, the initial simplicial complex is encoded in the winged data structure;
- for any other node U, the new vertex inserted by U is encoded (this defines an implicit description of the combination of the Node-Cell and the Cell-Vertex relations).

The cost of storing the implicit description is just equal to $\mathtt{v}$. It can be reduced to zero by storing vertices directly inside nodes. The total cost of this data structure is equal to $d\mathtt{v} + 4\mathtt{a}$, by considering the space required by the two DAG relations (i.e., NA and AN).

Given a front on the SMC, the vertex stored in a node U is sufficient to determine how the corresponding mesh must be updated when sweeping the front through U, either forward or backward. This operation reduces to vertex insertion or deletion in a Delaunay simplicial complex.

This compression scheme is easily implemented for 2-dimensional SMCs based on vertex insertions in a Delaunay triangulation. For higher values of the dimension d, the algorithms necessary to update the current Delaunay complex become more difficult [6]. Deleting a point from a Delaunay simplicial complex in three or higher dimensions, as required when sweeping backward the front, is not easy; we are not aware of any existing implemented algorithm for such task, even in the three-dimensional case.

A Structure Based on Edge Flips This compression scheme can encode any two-dimensional SMC where nodes represent vertex insertions in a triangle mesh. It is efficient for SMCs where the number of triangles created by each update is bounded by a small constant b. The basic idea is that, for each node U, the corresponding update (which transforms a triangle mesh not containing a vertex p into one containing p) can be performed by first inserting p in a greedy way and then performing a sequence of edge flips. This process, illustrated in Figure 7, defines an operational and implicit way of encoding the combination of the Node-Cell and Cell-Vertex relations.

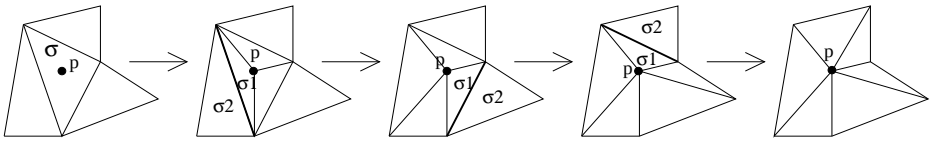

Fig. 7. Performing an update (insertion of a vertex p) through triangle split and edge flips. At each flip, the pair σ_1, σ_2 of triangle sharing the flipped edge is indicated.

First, p is inserted by splitting one of the triangles removed by p, that we call the *reference triangle* for p. This creates three triangles incident at p. Starting from such initial configuration, a sequence of edge flips is performed. Each edge flip deletes a triangle σ_1 incident at p, and the triangle σ_2 adjacent to σ_1 along the edge opposite to p, and replaces them with two new triangles incident at p, by flipping the edge common to σ_1 and σ_2. At the end, p has a fan of incident triangles which are exactly those introduced by update U.

The update represented by a node U is fully described by the new vertex p, a reference triangle σ, and a sequence of edge flips. Edge flips are represented by numerical codes. Let us consider an intermediate situation where a flip replaces the j-th incident triangle of p in a radial order around p (e.g., in counterclockwise order starting from the topmost triangle): then we use the number j to code the flip. The code of the first flip is in the range $0 \ldots 2$ since, at the beginning, there

are only three triangles incident at p. Since each edge flip increases the number of these triangles by one unit, the code for the j-th flip is in $0 \ldots j+1$. The total number of edge flips for a vertex p is $t-3$, where t is the number of triangles created by the update U. Since $t \leq b$ the flip sequence consists of at most $b-3$ integers, where the j-th integer is in the range $0 \ldots j+1$. Therefore, the sequence of flips can be packed in a *flip code* of $\sum_{j=1}^{b-3}(\log_2(j+2)) = \sum_{i=3}^{b-1}(\log_2(i)) = \log_2((b-1)!) - 1$ bits.

The reference triangle σ for p is a triangle created by some update U' that is a parent of U in the DAG. Thus, to uniquely define σ, it is sufficient to give a reference to U' and an integer number identifying one of the triangles incident in the central vertex of U' according to a radial order (e.g., counterclockwise). Conventionally, we organize the data structure in such a way that parent U' is the first parent stored for U; thus, there is no need to encode it explicitly. The number identifying σ is in the range $0 \ldots b-1$. We can pack the flip code and the index of σ together in $\log_2(b!) - 1$ bits. The space required for the implicit encoding of cells in this scheme is $\mathtt{v}((\log_2(b!) - 1))$ bits.

Extending this compression scheme to higher dimensions is a non-trivial task. Edelsbrunner and Shah [6] showed that insertion of a point in a Delaunay simplicial complex reduces to a sequence of flips of $(k-1)$-facets. This result could suggest that a coding based on flips may be possible for k-dimensional Delaunay SMCs built through incremental refinement. However, in k-dimensions it is not clear how the $(k-1)$-facets incident at a vertex could be sorted in such a way to allow the definition of a compact flip code. Moreover, it is difficult to guarantee a bounded degree of vertices in a k-dimensional simplicial mesh and, in any case, the number of flips is not guaranteed to be linear in the degree of the inserted vertex.

Discussion We have compared the sizes of the two compressed structures outlined above with the size of the explicit data structure, for a number of two-dimensional SMCs. On the average, the space occupied by the Delaunay compressed structure, and by the one based on edge flips, is about 1/4 and 1/3, respectively, of the space needed by the explicit structure without adjacencies.

It is interesting to compare the performance of query algorithms on an SMC when it is encoded through an explicit data structure, or through a compressed one, and the quality of the triangle meshes produced by such algorithms. Query algorithms provide the same meshes for the same input parameters with all compressed data structures, but the performances vary depending on the amount of work needed for reconstructing triangles with the specific structure.

Our experiments have shown that, if the given resolution filter does not refer to triangle attributes (e.g., it depends just on the geometry and location of triangles in space), the mesh extracted by a query algorithm using a compressed or an explicit structure are the same. On the contrary, if the resolution filter uses triangle attributes, then the resulting mesh may be quite different due to the approximation of such attributes in the compressed structures.

We have experimented with resolution filters that refer to *approximation errors* associated with triangles of an SMC. The resolution filter imposes an upper bound on the error of triangles that can be accepted in the solution of a query. In the compressed structure, a single error is associated with each node U, defined as the maximum approximation error of the triangles created by U. When such triangles are reconstructed in the current mesh, they receive the approximation error of U, which over-estimates their true error, hence forcing the extraction algorithm to over-refine the solution of the query. In this case, meshes extracted from the compressed SMC may be twice as large as those obtained from the explicit structure.

The performance of query algortihms has been monitored just for the explicit structure, and for the Delaunay-based compressed structure. The compressed structure based on edge flips is still under implementation. The explicit structure supports the extraction of triangle meshes formed by about 10^4 cells from SMCs containing about 10^5 cells, in real-time. Query algorithms on the Delaunay-based compressed structure are much slower. The increase in execution times is due to the on-line computation of a Delaunay triangulation. We expect better results with the structure based on edge flips.

6 Concluding Remarks

We have presented several alternative data structures for encoding a Simplicial Multi-Complex.

General-purpose data structures are characterized by encoding different subsets of the basic relations between the elements of an SMC. Different alternatives can be selected in order to adapt to the needs of a specific task, and to trade-off between space and performance. The SMC has been extended to cell complexes in [11]. However, the main difficulty in extending general-purpose data structures to general cell complexes lies in the intrinsic complexity of data structures for cell complexes, compared with those for simplicial complexes.

Compressed data structures have been defined for SMCs in the two-dimensional case, in which only the DAG structure is stored, and triangles are encoded through an implict rule. Only the structure for Delaunay SMC extends to three or more dimensions easily, even if the problem of deleting a point from a Delaunay mesh in three or higher dimension is solved only from a theoretical point of view. We plan to investigate more general compressed structures for higher-dimensional SMCs in the future.

Compressed data structures can be much more compact than general-purpose ones, but, on the other hand, the performance of extraction algorithms can be degraded severely, because of additional work necessary to reconstruct the structure of meshes. In the two-dimensional case, it should be remarked that, while the Delaunay-based data structure is more compact than the one based on edge flips, the performance of the extraction algorithms is severely affected by numerical computation necessary to update the Delaunay triangulation.

Based on the data structures presented here, we have developed an object-oriented library for building, manipulating and querying two-dimensional SMCs, which has been designed as an open-ended tool for developing applications that require advanced LOD features [11]. In the current state of development, the library implements both the explicit and the Delaunay-based data structures, the algorithms described in [11], algorithms for building an SMC both for terrains and free form surfaces, application-dependent operations implemented on top of the query operations, mainly for GIS applications (interactive terrain visualization, contour map extraction, visibility computations, etc.). We are currently implementing the structure based on edge flips and a version of the library for dealing with three-dimensional SMCs for representing 3D scalar fields at variable resolution.

An important issue in any application which deals with large data sets is designing effective strategies to use secondary storage. To this aim, we have been studying data structures for handling SMCs on secondary storage. In [11], a disk-based data structure for two-dimensional SMCs is proposed in the context of a terrain modeling application. Such a structure organizes a set of SMCs, each of which describes a subset of a larger area. Individual SMCs reside on separate files, and two or more of them (i.e., the ones contributing to represent a relevant area of space) can be merged into a single SMC when loaded into main memory to answer a query. Future work involves defining and implementing query algorithms having a direct access to large SMCs resident on disk.

Acknowledgments

Part of the work described in this paper has been developed while the first author was on leave from the University of Genova at the University of Maryland Institute for Applied Computer Studies (UMIACS). The support of National Science Foundation (NSF) Grant "The Grand Challenge" under contract BIR9318183 is gratefully acnowledged. This work has been also partially supported by the Coordinated Project "A Library for Applications in Geometric Modeling" of the Italian National Research Council under contract 98.00350.

References

1. M. de Berg and K. Dobrindt. On levels of detail in terrains. In *Proceedings 11th ACM Symposium on Computational Geometry*, pages C26–C27, Vancouver (Canada), 1995. ACM Press.
2. L. De Floriani, P. Magillo, and E. Puppo. VARIANT - processing and visualizing terrains at variable resolution. In *Proceedings 5th ACM Workshop on Advances in Geographic Information Systems*, Las Vegas, Nevada, 1997.
3. L. De Floriani, P. Magillo, and E. Puppo. Efficient implementation of multi-triangulations. In *Proceedings IEEE Visualization 98*, pages 43–50, Research Triangle Park, NC (USA), October 1998.
4. L. De Floriani and E. Puppo. Hierarchical triangulation for multiresolution surface description. *ACM Transactions on Graphics*, 14(4):363–411, October 1995.

5. L. De Floriani, E. Puppo, and P. Magillo. A formal approach to multiresolution modeling. In R. Klein, W. Straßer, and R. Rau, editors, *Geometric Modeling: Theory and Practice*. Springer-Verlag, 1997.
6. H. Edelsbrunner and N. R. Shah. Incremental topological flipping works for regular triangulations. *Algorithmica*, 15:223–241, 1996.
7. A. Guéziec, G. Taubin, F. Lazarus, and W. Horn. Simplicial maps for progressive transmission of polygonal surfaces. In *Proceeding ACM VRML98*, pages 25–31, 1998.
8. H. Hoppe. View-dependent refinement of progressive meshes. In *ACM Computer Graphics Proceedings, Annual Conference Series, (SIGGRAPH '97)*, pages 189–198, 1997.
9. P. Lindstrom, D. Koller, W. Ribarsky, L.F. Hodges, N. Faust, and G.A. Turner. Real-time, continuous level of detail rendering of height fields. In *Comp. Graph. Proc., Annual Conf. Series (SIGGRAPH '96), ACM Press*, pages 109–118, New Orleans, LA, USA, Aug. 6-8 1996.
10. D. Luebke and C. Erikson. View-dependent simplification of arbitrary polygonal environments. In *ACM Computer Graphics Proceedings, Annual Conference Series, (SIGGRAPH '97)*, pages 199–207, 1997.
11. P. Magillo. *Spatial Operations on Multiresolution Cell Complexes*. PhD thesis, Dept. of Computer and Information Sciences, University of Genova (Italy), 1999.
12. A. Paoluzzi, F. Bernardini, C. Cattani, and V. Ferrucci. Dimension-independent modeling with simplicial complexes. *ACM Transactions on Graphics*, 12(1):56–102, January 1993.
13. E. Puppo. Variable resolution terrain surfaces. In *Proceedings Eight Canadian Conference on Computational Geometry*, pages 202–210, Ottawa, Canada, August 12-15 1996.
14. E. Puppo. Variable resolution triangulations. *Computational Geometry Theory and Applications*, 11(3-4):219–238, December 1998.
15. J.C. Xia, J. El-Sana, and A. Varshney. Adaptive real-time level-of-detail-based rendering for polygonal models. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):171–183, 1997.