A topological data structure for hierarchical planar subdivisions*

Waldemar Celes Filho[1] Luiz Henrique de Figueiredo[2] Marcelo Gattass[3]
Paulo Cezar Carvalho[4]

Abstract

We introduce HPS, a new topological data structure that efficiently represents hierarchies of planar subdivisions, thus providing direct and efficient support for GIS concepts such as abstract generalizations and multi-scale partitions. Unlike previous ad hoc solutions, HPS provides efficient access to adjacency information for each level and across levels, while storing the complete hierarchy in a single data structure, without duplications. HPS also provides topological operators that ensure global consistency. Like all topological data structures, HPS can be used as a framework onto which geometric and attribute information is placed: HPS explicitly handles attributes consistently with modeling, and naturally supports both topological and geometrical multi-resolution representations. We also discuss how some typical applications in GIS, Digital Cartography, and Finite Element mesh generation can be improved with HPS.

KEYWORDS: topological data structures, hierarchical modeling, multi-resolution, multi-scale partitions, Geographic Information Systems.

1 Introduction

The maps used in Geographic Information Systems (GIS) and Digital Cartography are abstract models of real geographic data that help the extraction of all sorts of metric and topological information. The two main types of maps are *reference maps* and *thematic maps*. Reference maps represent a precise model of reality and require high positional accuracy; examples are cadastral maps, and highway or water distribution networks. Thematic maps handle different aspects of a single theme of interest, such as weather forecast and demographic distribution; they can trade positional accuracy for artistic enhancement, and may be caricatural in their design [1].

There are two major representations for maps: *raster* and *vector*. Although both approaches are valid, and may co-exist in practice [2], the requirements of reference maps are not supported by

*Technical report CS-95-53, Dept. of Computer Science, U. of Waterloo. Submitted to *Algorithmica*. Research done at TeCGraf, PUC–Rio, Brazil, and partially supported by the Brazilian Council for Scientific and Technological Development (CNPq), including its PROTEM GEOTEC project, and the Brazilian Oil Company (PETROBRAS).

[1]Program of Computer Graphics, Cornell University, 580 Engineering Theory Center Building, Ithaca, NY 14853. (celes@graphics.cornell.edu) Holder of a post-doctoral fellowship from CNPq.

[2]Computer Systems Group, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1. (lhf@csg.uwaterloo.ca) Holder of a post-doctoral fellowship from CNPq.

[3]TeCGraf, Computer Science Department, PUC–Rio, Rua Marquês de São Vicente 225, 22453-900 Rio de Janeiro, RJ, Brazil. (gattass@icad.puc-rio.br)

[4]IMPA–Instituto de Matemática Pura e Aplicada, Estrada Dona Castorina 110, 22460-320 Rio de Janeiro, RJ, Brazil. (pcezar@visgraf.impa.br)

either raster or unstructured vector representations (also known as "spaghetti" [3]). Analysis of reference maps frequently requires topological information, i.e., adjacency information about primitive map entities. Moreover, many operations, such as computing network connectivity, areal extent and adjacency, and even polygon coloring in thematic maps, can be performed more efficiently if adjacency relationships are known explicitly [3].

A good mathematical model for a map is a subdivision of the plane into regions, also called a *planar subdivision*. Data structures for faithfully representing planar subdivisions, keeping track of all adjacency relationships, are called *topological data structures*. Although it is clear that such data structures are useful in GIS and related areas, many applications actually deal simultaneously with several, hierarchically related planar subdivisions. For instance, cartographic maps contain several levels of subdivision (countries, states, counties, etc.), and each level is a refinement of the preceding one; cadastral applications may represent city-blocks, parcels, and buildings in different levels to improve data structure organization. Thus, many important applications require not only explicit topological information, but also support for hierarchies.

It is possible to use a conventional topological data structure [4, 5, 6] to represent a hierarchy of planar subdivisions. However, it is conceptually cleaner to use data structures that directly support hierarchical relationships; it is also much easier to maintain global consistency. In this paper, we present HPS, a new topological data structure with built-in support for hierarchical planar subdivisions. HPS represents the complete multi-level model in a single data structure, and provides a set of basic topological operators to ensure model consistency, avoiding duplications of identical entities. By directly representing hierarchical relationships, we avoid ad hoc solutions such as grouping features according to common attributes. For example, we can directly represent states as part of countries, without having to "tag" all states with their respective countries. Maintaining such multiply layered tags in actual hierarchies can be very complicated, and is just one of the problems of using conventional topological data structures to represent hierarchies.

For completeness, we start by describing, in Section 2, planar subdivisions and a class of topological data structures for representing them. This section also serves as an introduction for the fundamental concepts behind HPS. In Section 3, we discuss how conventional topological data structures can be used in an ad hoc way to represent hierarchies of planar subdivisions. We also discuss the requirements for efficiently representing hierarchical planar subdivisions, and give a formal definition of consistency in hierarchical planar subdivisions. In Section 4, we introduce the HPS data structure, which satisfies these requirements, and describe its features and implementation. In Section 5, we describe some applications that can be improved by explicitly representing hierarchies. Section 6 contains discussions and suggestions for further research.

2 Representing planar subdivisions

A graph is *planar* if it can be embedded in the plane without crossings. A *planar subdivision* is a partition of the plane determined by an embedded planar graph [7]. A planar subdivision contains three kinds of topological entities: vertices, edges, and faces. A *vertex* is a 0-dimensional entity and represents a single point on the plane. An *edge* is a 1-dimensional entity and connects two, not necessarily distinct, vertices. A *face* is a 2-dimensional entity and represents a connected subset of the plane, possibly with holes.

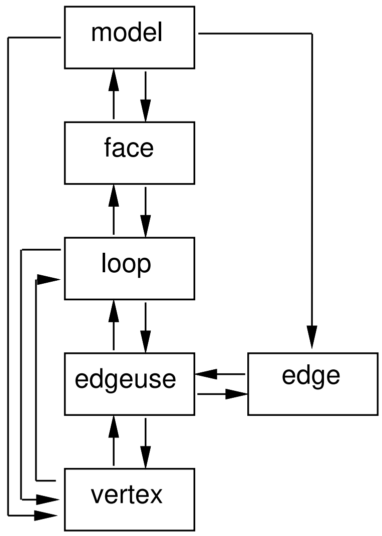

Figure 1: Top-down organization of an edge-based topological data structure for representing planar subdivisions.

The fundamental topological properties characterizing planar subdivisions are [4]:

- faces may intersect only at common edges or vertices;
- each edge is shared by exactly two faces;
- faces around each vertex form a single circuit, i.e., they can be rearranged in a cycle such that each pair of consecutive faces in the cycle meet at an edge adjacent to the vertex.

Topological data structures for representing planar subdivisions store sufficient information to allow efficient extraction of all adjacency relationships among vertices, edges, and faces. Many edges can be adjacent to a vertex or an edge, many faces can be adjacent to a vertex and vice-versa, but each edge is adjacent to exactly two faces and two vertices. Thus, it is natural to base representations for planar subdivisions on edge adjacencies.

The first edge-based data structure was the *winged-edge* data structure [8]. Many other data structures have been proposed since then (e.g., *half edge* [4], *face-edge* [5], *quad-edge* [6]), but they are all variations of the winged-edge. Of interest here are variants that explicitly support orientation and faces with holes. Besides vertices, edges, and faces, these variants explicitly represent loops and edge uses. *Loops* provide a representation for faces with holes. Each face has several loops: one loop represents the external boundary; the others represent internal boundaries, corresponding to holes within the face. *Edgeuses* correspond to each use of an edge by a face. An edge has exactly two edgeuses; each edgeuse is adjacent to exactly one face. Edgeuses allow the representation of a globally consistent orientation of the faces because edgeuses are oriented, even though edges are not. This orientation is useful in many algorithms.

Figure 1 shows the top-down organization of an edge-based data structure containing loops and edgeuses. This organization is useful in many modeling applications because objects can often

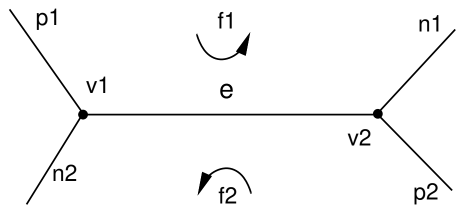

Figure 2: The complete "winged-edge" of an edge.

be processed at higher levels of abstraction that roughly correspond to grosser geometric features [9]. Such a data structure supports the representation of several independent models. A *model* represents a complete planar subdivision. Within each model, one can access the faces, edges, and vertices that realize the representation. A face is represented by a list of loops; each loop is an ordered cycle of edgeuses (or possibly a single vertex). An edge has two edgeuses; each edgeuse has references to its edge, to its start vertex (recall that edgeuses are oriented), and to its adjacent loop. Edgeuses also keep references to the next and previous uses within the adjacent loop. With these references, one can extract the complete "winged-edge" of an edge: its two end vertices, its two adjacent faces, and the edges that precede and succeed the edge in its adjacent loops (Figure 2). Finally, a vertex has a reference to an adjacent edgeuse, or to a loop, if the vertex is isolated.

2.1 Adjacency relationships

Weiler [5, 9] defines *sufficiency* of a data structure as the ability to recreate all adjacency relationships among vertices, edges, and faces, without error or ambiguity. He proved the sufficiency of his variant of the winged-edge data structure. There are nine adjacency relationships:

- VV: given a vertex, find the ordered cycle of vertices that are adjacent to it. Two vertices are adjacent if they are connected by an edge.
- VE: given a vertex, find the ordered cycle of edges that are adjacent to it. An edge is adjacent to a vertex if the vertex is an extreme of the edge.
- VF: given a vertex, find the ordered cycle of faces that are adjacent to it. A face is adjacent to a vertex if the face contains the vertex as a single loop or shares at least one of the edges adjacent to the vertex.
- EV: given an edge, find the two vertices connected by the edge.
- EE: given an edge, find the edges adjacent to each one of its adjacent vertices.
- EF: given an edge, find the two faces that share it.
- FV: given a face, find the ordered cycles of vertices representing the loops of the face.
- FE: given a face, find the ordered cycles of edges representing the loops of the face.
- FF: given a face, find the set of faces adjacent to it. Two faces are adjacent if they share an edge.

These adjacency relationships can all be obtained in time proportional to the size of the answer, with no geometric processing. However, real geographical maps are rarely general planar subdivisions; most likely, for instance, a vertex has only few adjacent edges (in a map of the United States, all vertices have three adjacent edges, except for one that lies at the common border of Utah, Colorado, Arizona, and New Mexico). Thus, in representations of actual maps, all adjacency relationships can probably be obtained in constant time.

2.2 Euler operators

Euler operators, also originally presented by Baumgart [8], effectively hide the details of the representation and provide a stepwise fashion to construct the model. The set of Euler operators vary slightly across implementations. The set we use here contains the operators below and their inverses:

- MVFM: make vertex, face, and model;
- MVR: make vertex and ring;
- MVSE: make vertex, split edge;
- MEKR: make edge, kill ring;
- MESF: make edge, split face.

This and other equivalent sets of Euler operators are sufficient to create any planar subdivision, and their use does not create topologically inconsistent models [4]. Thus, Euler operators not only hide implementation details, but also guarantee consistency. It is worth noticing that any model can be built with a sequence of Euler operators; moreover, given the model, this set of operators can be easily found [10].

Building models with topological operators also allows client applications to easily manage non-graphical properties (attributes): it is enough to notify the client every time an operator is executed; the client can register a function to handle such notifications (this is how HPS handles attributes). However, this approach is only useful if each operator has a clearly defined semantics, because client responses to different operators can be completely different. The set of Euler operators listed above was chosen based on this principle. Thus, for instance, splitting an edge (with MVSE) can cause the client to refine the attributes of the split edge into attributes for the two edges being created. On the other hand, an edge created with MEKR or MESF can simply receive current attributes. As an example of an operator without clearly defined semantics, take the operator MEV ("make edge and vertex") [11], which is not in our set. This operator is used both to split an existing edge and to create a new edge and vertex within a face; it does not have a clearly defined semantics because clients would not be able to decide how to handle attributes in this case.

3 Representing hierarchies of planar subdivisions

Once topological data structures are available for representing planar subdivisions, it is clear that they can be used for representing hierarchies too. In this approach [12], each hierarchical level is stored in a separate data structure, and ad hoc techniques are used to provide support for hierarchical relationships between consecutive levels. Therefore, although horizontal consistency (i.e., at each level) can be guaranteed by Euler operators, vertical consistency (i.e., across levels) cannot be guaranteed due to the ad hoc nature of inter-level linking. Moreover, code for applications designed with this approach tends to be exceedingly complex and hard to maintain, since supporting multiple subdivisions requires manipulating very complex data structures at a relatively low level. Another disadvantage of this approach is the waste of space due to the replication of entities that are identical in several levels.

A data structure with built-in support for hierarchies needs to provide:

- efficient access to horizontal and vertical adjacency information;
- topological operators to ensure horizontal and vertical consistency;
- a single data structure to hold the complete model, avoiding duplication.

To design such a data structure, it is beneficial to define formally what hierarchical planar subdivisions are, thus clearly delimiting a domain for the representation. In this paper, a *hierarchy* is a sequence of *levels*; each level contains a single valid planar subdivision. The topological elements contained in a hierarchical planar subdivision are the usual ones: vertices, edges, and faces. However, they must satisfy not only the requirements that define horizontal consistency listed in Section 2, but also two additional requirements that define vertical consistency:

- *top-down consistency*: each entity x is entirely contained in some entity y at each level above it. We say that x is *shadowed* by y;
- *bottom-up consistency*: each entity x is the union of entities at each level below it. We say that x is *refined* at the lower levels.

With this definition of vertical consistency, we are restricted to representing a hierarchy of partitions of a single geographical space, also called *multi-scale partitions* [13]. However, this restriction is commonly satisfied in practice.

Figure 3 illustrates a simple hierarchical model with two levels. (In this paper, figures display hierarchies horizontally, with top levels on the left and bottom levels on the right.) Face f_2 of the top level is *refined* into faces $\{f_3, f_4\}$ of the bottom level. Similarly, edges e_1 and e_3 are refined into $\{e_5, e_6\}$ and $\{e_7, e_8\}$, respectively. On the other hand, vertices v_5 and v_6, and edge e_9 are "new" entities of the bottom level, but are *shadowed*: v_5 is shadowed by e_1, v_6 is shadowed by e_3, and e_9 is shadowed by f_2. Indeed, e_1 is the union of $\{e_5, e_6, v_5\}$; e_3 is the union of $\{e_7, e_8, v_6\}$; and f_2 is the union of $\{f_3, f_4, e_9\}$.

4 The HPS data structure

The design goal of HPS is to represent, in a single data structure, the complete multi-level model, with efficient access to horizontal and vertical adjacency information, and without duplicating entities. The HPS data structure is an extension of the edge-based topological data structures described in Section 2, and its design was inspired in the work of Martha [12] (who used hierarchical

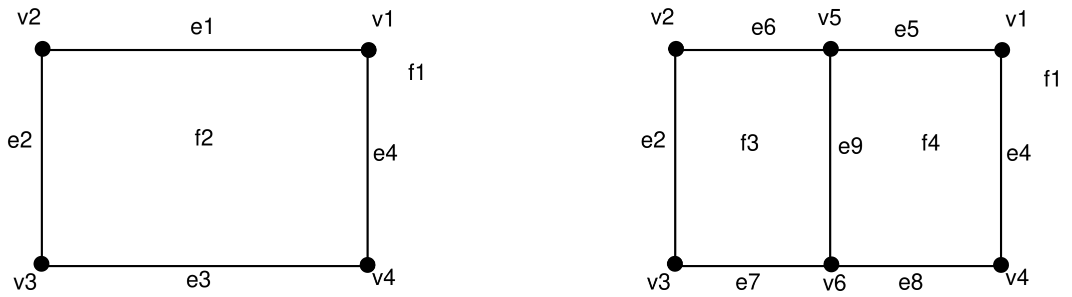

Figure 3: A simple two-level hierarchical model.

planar subdivisions to model finite-element meshes; see Section 5). Although we only address here the two-dimensional case, the approach described here can be extended to three dimensions, by extending, for instance, the *radial edge* data structure [5].

4.1 Avoiding entity duplication

In HPS, topological entities are represented at the level that they appear, without being duplicated at lower levels. To avoid such duplication, we slightly restrict the domain defined in Section 3, requiring that *each entity of a given level, when not subdivided at the level below, also belongs to this level.* Thus an entity is *defined* at a unique level, but may also belong to several, consecutive lower levels. The level at which the entity first appears (from top to bottom) defines the entity. Therefore, lower levels can include entities from higher levels, with no duplications.

An alternative approach would be to let the client (i.e., the application program) decide when to allow entity duplication. Although apparently more flexible, this approach places an extra burden on the client: to decide which entities must be duplicated, and when to do so. In practice, the domain restriction made by HPS does not affect applications and provides high-level control over semantic consistency. For instance, consider a cartographic map where a state has a single county. In such a map, two different semantic entities (state and county) may be represented by a single topological entity (a face). One can claim that this is ambiguous because we may need to attach different attributes (non-graphic properties) to them. However, we argue that in such situations either the attributes are of different classes (and then they do not conflict with each other), or the attributes are of the same class (and then should be assigned only once).

4.2 Top-down organization

Since HPS is an extension of conventional edge-based topological data structures, its organization is very similar to that described in Section 2. Topological entities are organized in a top-down structure, where higher dimensional entities are supported by lower dimensional entities. This structure is essentially the one shown in Figure 1, except for three major changes: First, faces and edges are kept in *trees*, rather than lists. This is necessary to represent faces or edges that are refined at lower levels. Vertices are still stored in lists because they cannot be refined. Second, edgeuses have many additional links, two (*next* and *previous*) for each level below their definition level, and hence are kept in *multilink lists.* Finally, a vertex has a reference to an adjacent entity,

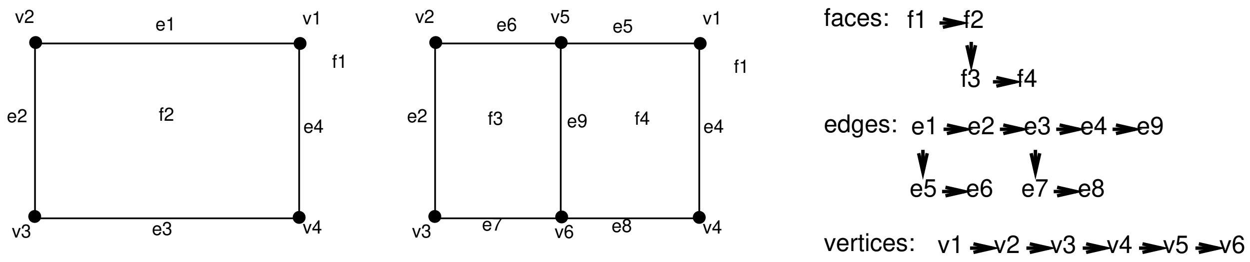

Figure 4: The simple two-level hierarchical model represented with HPS.

which can be an edgeuse in the lowest level or a loop in the lowest level, if the vertex is isolated in this level. Note that a vertex belongs to all levels below its definition level, because it cannot be refined; however, HPS stores just that single reference to an adjacent edgeuse or loop in the lowest level, because we can obtain references to adjacent edgeuses or loops in other levels by using vertical adjacency information, as described in Section 4.3.

Figure 4 shows the simple hierarchical model shown Figure 3 and how its faces, edges, and vertices are organized. To see why multilinks are needed in edgeuses, consider traversing all edges adjacent to face f_2 in the top level. During this traversal, edge e_2 is followed by edge e_3 (considering a counterclockwise orientation of f_2). In a similar traversal of f_3 in the bottom level, e_2 is now followed by e_7. Therefore, the edgeuse associated with e_2 must have two links: one associated with edge e_3, used for traversal in the top level; and one associated with edge e_7, used for traversal in the bottom level. Note also that the external face also belongs to the bottom level; therefore, its adjacent edgeuses, of both levels, have a reference to the same loop in the external face. It is in this sense that HPS avoids entity duplication. Finally, note that vertex v_1 has a reference to a use of either e_4 or e_5, which are in the bottom level, but v_1 does not have a reference to a use of e_1.

4.3 Adjacency relationships

HPS provides easy access to topological information and efficiently gives all adjacency relationships, which are now of two types: non-hierarchical and hierarchical. Non-hierarchical relationships are the ordinary adjacency relationships among faces, edges, and vertices at a single hierarchical level. Hierarchical relationships relate topological entities across different hierarchical levels. We define them here as the ability to provide *parent/children* information about edges and faces. So, besides the relationships listed in Section 2.1, we have EE^-, FF^-, EE^+, FF^+ (given an entity, find its parent or its children).

Hierarchical relationships are directly provided by HPS because of its tree structures for edges and faces. Non-hierarchical relationships are obtained efficiently because, when restricted to a single level, HPS is equivalent to a conventional edge-based topological data structure for representing the planar subdivision at this level. However, since a face may belong to more than one level, obtaining relationships VF and EF may require a local search. This happens because we may need to find an adjacent loop among the loops of a parent or a set of children faces [14]. Nevertheless, in real applications, all adjacent relationships are obtained efficiently due to the limited, and relatively small, number of loops of a face (and number of children of a face), as mentioned in Section 2.1.

4.4 Topological operators

To guarantee topological consistency throughout the editing process, HPS provides a convenient set of basic topological operators that, like ordinary Euler operators, are responsible for all changes in the data structure. The operators provided by HPS are essentially Euler operators, because editing a hierarchical planar subdivision is very similar to editing a single planar subdivision. The difference is that the topological operators in HPS may act on several levels simultaneously. For instance, a vertex inserted at the top-most level will also be present in all lower levels. Therefore, the topological operators in HPS are called *extended Euler operators*, but have the same names and semantics (extended for hierarchies) of those listed in Section 2.2.

The extended action of each operator is determined by the level of the entity on which it acts. If an edge is inserted in such a way as to split a face that has children at a lower level, then insertion occurs automatically in a set of adjacent levels, from the level of the face being split to the level immediately above that of the children. Observe that, without violating consistency, such an insertion is possible only if at the level of the children there are edges that completely cover the edge being inserted. If necessary, this policy can be overruled by high-level operators supplied by the application; such operators can adjust lower levels to accommodate inconsistencies due to insertions at a higher level. For instance, if an edge inserted at a higher level crosses existing edges at lower level, then a high-level operator can either insert the new edge also at the lower level or remove the edges at the lower level that cross the new edge, because the previous refinement can be no longer valid.

HPS also provides three auxiliary operators (and their inverses):

- CF: copy face;
- CE: copy edge;
- CEP: change edge parent.

The operator CF copies a face from a higher level to a given level. This operator is needed for inserting a vertex or an edge on a face that is defined at a higher level, because insertions cannot be performed across levels. The operator CE is similar to CF, but acts on edges. The operator CEP is needed when an edge is inserted over lower edges, covering them. In this case, the lower level edges become children of the new edge. This could happen when a new state is created and the boundary of this state coincides with existing county boundaries. (This happened recently when the Tocantins state was created in Brazil, and similarly when new countries were created as the result of political reorganization of Europe.)

To illustrate the action of the basic operators, take the simple three-level model in Figure 5. Note that the internal face of the top level also belongs to the middle level. Consider the insertion of an edge from vertex v_i to v_j at the top level. This can be achieved by single use of the operator MESF, which acts on the top two levels simultaneously, followed by three uses of CEP to fix edge parents on the bottom level. Now, consider the insertion of the same edge, but at the middle level instead of at top level. Before splitting the face, we need to copy the top face to the middle level (using CF), and then perform the splitting in the usual way with MESF. Note that this time MESF acts only on the middle level.

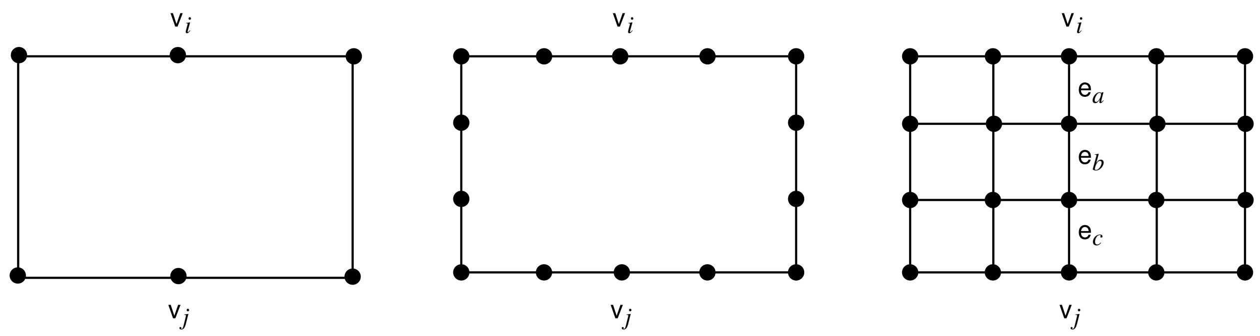

Figure 5: A simple three-level hierarchical model.

Strictly speaking, the auxiliary operators are not *operators* since, if applied independently, they can render the data structure into a state outside the intended domain of the representation. For instance, applying a single CF operator would create face duplication, and this is not allowed. However, the auxiliary operators do provide a desirable modularity to the implementation of a data structure for representing hierarchical planar subdivisions, allowing it to start from an existing implementation of a conventional topological data structure and its Euler operators.

5 Some potential applications

Several applications that deal with planar subdivisions are intrinsically hierarchical. Using a data structure that has built-in support for multi-level models is the easiest and most appropriate way to implement such applications. In this section, we discuss some applications that can be improved if built upon the topological framework provided by a data structure such as HPS.

5.1 Political cartographic maps

The classical example of multi-level maps is the representation of the political distribution of countries, states, counties, etc. Such data is naturally organized as a hierarchy of planar subdivisions: Higher level semantic entities (e.g., countries) impose boundary restrictions to, and are subdivided by, lower level semantic entities (e.g., states). Data at lower topological levels has higher semantic level, because the hierarchy is used for abstract generalization [13].

HPS provides an adequate framework to represent such maps. It is natural to choose to represent each set of semantic entities (countries, states, counties, etc.) at a different level. Besides this natural and well-structured organization, the use of HPS in such applications avoids the duplication of geometric information: lower entity boundaries (e.g., polygonal descriptions) either share higher entity boundaries or are described by a subset of these higher boundaries. Figure 6 gives an example of this application. The total number of topological entities in these maps is 1029; by avoiding duplications, HPS represents the complete hierarchy using only 405 topological entities.

Figure 6: A country (Brazil) (left), subdivided into 5 administrative regions (middle), which are subdivided into 26 states (right).

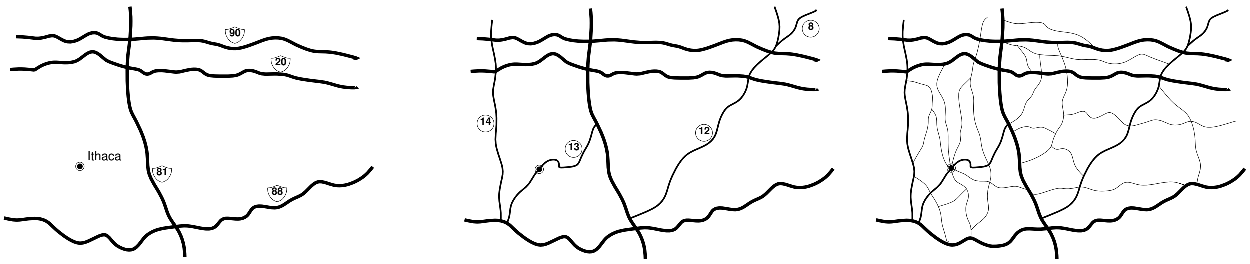

Figure 7: A multi-level model of the roads in the vicinity of Ithaca, NY: Interstate and Federal Highways (left), State and Provincial Highways (middle), County and Local Highways (right).

5.2 Road atlases

The representation of network maps such as road atlases is another example that is naturally supported by HPS. To perform analysis on a network representation, we need connectivity information [15]. For instance, consider the task of route planning upon a road network: The goal is to find out the best way to connect two points, satisfying predefined constraints. A simple, but inefficient, way to accomplish this task is to represent the entire network using a conventional topological data structure. The drawback of this approach is that the topological data structure, having to deal with nodes and links, can disrupt the smoothness of entity representation: a long highway may be fragmented due to secondary highway junctions [3]. To overcome this problem, we propose a hierarchical organization of such networks and design a multi-level model with HPS: Interstate and Federal Highways in the top level, State and Provincial Highways in the middle level, and County and Local Highways in the bottom level (Figure 7). With this strategy, we can perform macro-analysis considering only Interstate and Federal Highways, stored on the top level without fragmentation; or we can perform detailed analysis considering the levels below. Note that geometric information is shared by all three levels.

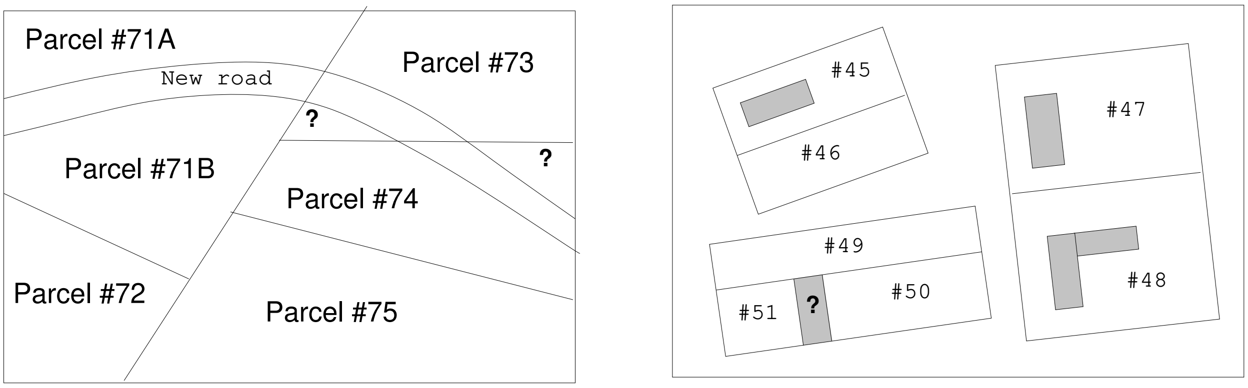

Figure 8: Ambiguities in a cadastre [15]: residual small plots of land due to a new road (left); the parcel containing the marked building can be either #50 or #51 (right).

5.3 Cadastral maps

Maps are central to many land- and property-based information systems, which are concerned with the factual information about locations and places [1]. The goal is to identify the several buildings belonging to a given parcel and vice-versa [15]. Again, topological information about areal properties is needed [15]. Also, like the previous applications, these applications have a natural hierarchical structure. Such maps usually represent three semantic entities: city-blocks, parcels, and buildings. With a conventional topological data structure, several ambiguities may appear: A new road may subdivide a parcel creating residual small plots of land; or if a building is located on the boundary of a parcel, then the parcel containing the building is not well defined (Figure 8).

To solve these ambiguities, one can use artificial entities, such as arrow membership [15]. However, a multi-level representation for cadastral maps naturally eliminates these ambiguities: It avoids residual plots of land that do not appear as actual parcels since one can represent parcel at one level and roads at a level below; it also avoids ambiguous parcel-building connections since we can see building as a refinement of parcels by representing them at different levels. Figure 9 illustrates this solution.

5.4 Finite element meshes

Besides GIS and Digital Cartography, many other applications in Engineering, Mining, and Geology also require the representation of hierarchical planar subdivisions. Although there are major differences between GIS and CAD applications, the topology in both cases may well be very similar [16]; it is no accident that topological data structures were originally designed for modeling solids [8]. In this section, we discuss the use of HPS for representing finite element meshes. In fact, HPS was originally designed to support multi-level models in finite element mesh generation.

The finite element method for performing mechanical analysis requires domain discretization, i.e., the entire domain must be subdivided into small regions (which are the "finite elements"). Several automatic methods for generating such discretizations impose additional topological re-

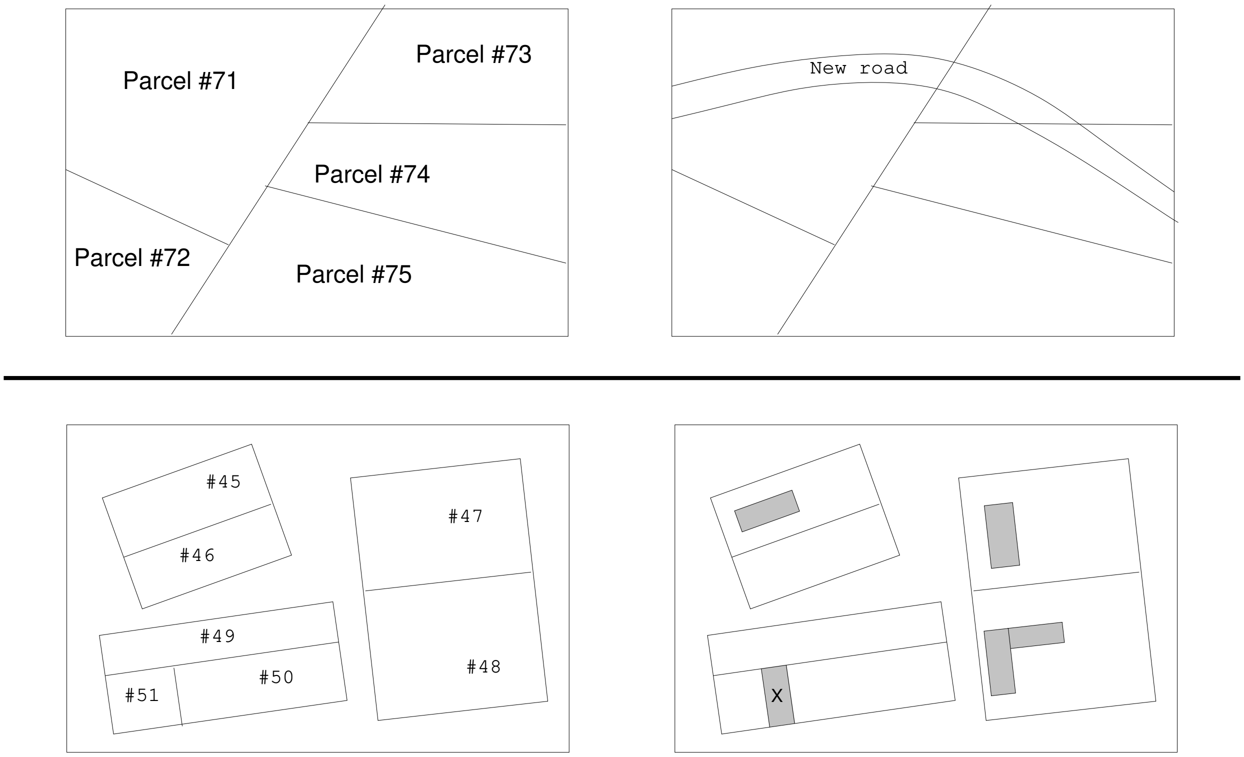

Figure 9: Multi-level models to solve ambiguities in the cadastres of Figure 8: no residual plots of land due to new road (top); the marked building is in parcel #51, not in #50 (bottom).

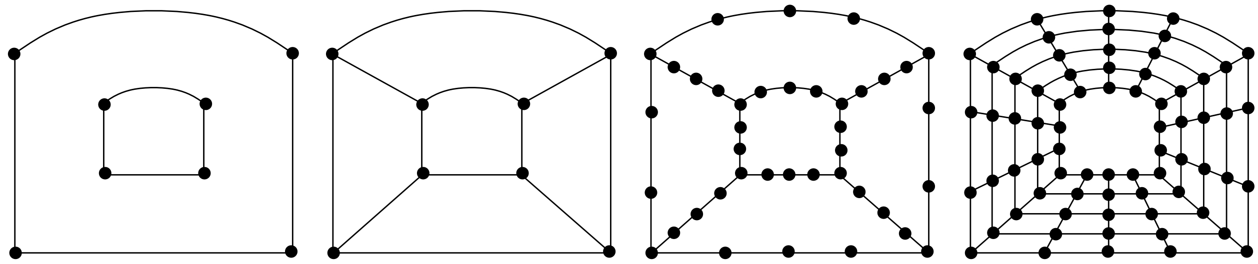

Figure 10: A multi-level model of a finite element mesh: domain geometry, region subdivisions, boundary discretizations, final mesh.

strictions on element meshes. For instance, transfinite mapping can only be applied over regions that are topologically equivalent to rectangles or triangles (i.e., to faces with three or four vertices). Due to these topological restriction, and also to the need of representing different non-graphical properties (attributes), Martha [12] designed a hierarchical multi-level representation for the entire domain. The top level represents the domain geometry; the second level represents the region subdivisions needed to apply automatic mesh generation methods; the third level represents boundary discretizations; and the bottom level represents the complete mesh (Figure 10).

This multi-level representation for meshes is, not surprisingly, naturally supported by HPS. In Figure 10, the third level is composed only by entities that also belong to other levels. This level has an important meaning for the application, but its representation in HPS does not imply an increase of the number of entities in the whole model. This fact illustrates an important feature of HPS: semantic levels can be created as needed to better describe a multi-level model, without increasing the number of represented entities.

6 Discussion

The main advantage of topological data structures is that, by design, they provide efficient access to topological information. Topological data provide information in a systematic and complete manner, with no need to consider positional information [2]. Moreover, topological data structures can be used as a framework onto which geometric and attribute information is placed [17]. Geometric information is specially useful for edges. Edges are *topological* elements, and do not have to be straight lines segments; edges can be polygonal lines, circular arcs, or any parametric curve. Any curve representation can be used, as long as it does not violate topological consistency.

6.1 Objections to topological data structures

Topological data structures have drawbacks that must be addressed. One is the volume of data required, which is large when compared to spaghetti data. However, in GIS and cartography, geometrical information is many orders of magnitude larger than topological information. For example, in the cartographic map shown in Figure 6, the number of topological entities is relatively small (a total of 405 vertices, edges, and faces in all three levels), but 24,306 points are needed to represent the polygonal geometry of edges.

Another disadvantage of topological data structures is the fragmentation of continuous data: As described in Section 5.2, having to deal with nodes and links can disrupt the smoothness of a entire entity. For instance, a edge representing a highway is broken into several nodes due to secondary road intersections. However, the use of a hierarchical model, represented by a topological data structure such as HPS, easily overcomes this problem, and provides an effective way to implement such applications.

Finally, topological data structures are typically implemented with a web of pointers to dynamically allocated memory. Because no special care is taken to guarantee any kind of reference locality, handling very large models can be inefficient in memory paging environments. This issue can be specially important in GIS applications, but does not seem to have been addressed.

6.2 Hierarchies and multi-resolution

The word *hierarchy* can have different meanings in GIS. A common meaning is related to the type of data structure, e.g., quadtree or octree [18]. In this case, the data structure is hierarchically organized, but not necessarily the model. We use a different meaning for *hierarchy* in this paper. The models we aim to represent are themselves hierarchical, i.e., they are composed of several levels, arranged into a hierarchy. Each level has its own semantics, although it can also be seen as a refinement of the preceding level. HPS was designed to represent such hierarchical models in a consistent way. Thus, HPS addresses a concern expressed by Egenhofer [19]: "a major impediment in the transition to more powerful multiple-representation GIS is the lack of methods to maintain consistently the multiple representation of geographic objects".

Another common use of *hierarchy* is in the context of multi-resolution maps [20, 21]. In this case, the goal is to represent the same map at different resolutions. A multi-resolution representation uses a different number of entities for each scale, thus providing an efficient way to draw maps at any scale. For instance, in the representation of a cartographic map, the boundary of a state can be represented by topological entities of different levels, varying according to the resolution [20], while remaining a single semantic entity. This kind of multi-resolution is called *topological multi-resolution*, because the topology of the model varies according to the scale. For example, at low resolution, a face may be represented as a single vertex. Topological multi-resolution can be implemented with HPS, provided the restriction made in Section 4.1 is satisfied. Thus, to represent a face as a single vertex at low resolution, this vertex must be part of the face. However, as mentioned before, HPS faithfully represents *multi-scale partitions* [13], a common case of topological multi-resolution.

A similar, but different, kind of multi-resolution is *geometric multi-resolution*: The complexity of the geometry varies according to the scale, and this also provides an efficient way to draw maps at any scale. If edges have polygonal geometry, then strip trees [22] can be used to implement geometric multi-resolution. If edges have curved geometry, then arc trees [23] can be used. In any case, geometrical multi-resolution and topological multi-resolution are actually orthogonal concepts. For instance, non-hierarchical planar subdivisions can use strip trees for efficient rendering according to display resolution and zoom factor. Nevertheless, it is simple to integrate geometric multi-resolution into a hierarchical data structure such as HPS because geometric refinement can be naturally represented in parallel to topological refinement, without duplicating geometric information. Our future plans include implementing geometric multi-resolution and overlay algorithms using HPS.

6.3 Conclusion

Although topological data structures are more complex to implement and maintain, they are now well-understood and provide very useful additional information, while guaranteeing the consistency of models. The HPS data structure introduced here is a relatively simple extension of conventional topological data structures that efficiently represents hierarchies of planar subdivisions, naturally catering for the needs of GIS and Digital Cartography. Moreover, a data structure with built-in support for multi-level models is the recommended tool for implementing applications that are intrinsically hierarchical.

Due to the importance of historical information in GIS applications [2, 16, 24], it is worth noting that the topological operators in HPS provide a effective way to describe the model. Because it is easy to enumerate the set of operators necessary to reconstruct a given configuration, it may be useful to investigate operator enumeration strategies to store historical information. We also plan to investigate efficient ways to store and retrieve topological information using general purpose database systems.

References

[1] M. Visvalingam. Trends and concerns in digital cartography. *Computer-Aided Design*, 22(3):115–130, April 1990.

[2] N. S. Smith. Spatial data models and data structures. *Computer-Aided Design*, 22(3):184–190, April 1990.

[3] R. Laurini and D. Thompson. *Fundamentals of Spatial Information Systems.* Academic Press, 1992.

[4] M. Mäntylä. A note on the modeling space of euler operators. *Computer Vision, Graphics and Image Processing*, 26:45–60, 1984.

[5] K. Weiler. *Topological structures for geometric modeling.* Ph.D. thesis, Rensselaer Polytechnic Institute, August 1986.

[6] L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivision and the computation of Voronoi diagrams. *ACM Trans. on Graphics*, 4:74–123, 1985.

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

[8] B. Baumgart. Winged–edge polyhedron representation. Technical Report CS–320, Stanford University, Stanford, CA, 1972.

[9] K. Weiler. Edge-based data structures for solid modeling in curved-surface environments. *IEEE Computer Graphics and Applications*, 5(1):21–40, 1985.

[10] M. Mäntylä. An inversion algorithm for geometric models. *Computer Graphics (SIGGRAPH '82 Proceedings)*, 16(3):51–59, July 1982.

[11] M. Mäntylä. *An Introduction to Solid Modeling.* Computer Science Press, Rockville, Md, 1988.

[12] L. F. Martha. *Topological and geometrical modeling approach to numerical discretization and arbitrary fracture simulation in three-dimensions.* Ph.D. thesis, Cornell University, 1989.

[13] P. Rigaux and M. Scholl. Multi-scale partitions: Application to spatial and statistical databases. In *Advances in Spatial Databases (Proc. of SSD'95)*, volume 951 of *Lecture Notes in Computer Science*, pages 172–183. Springer, 1995.

[14] W. Celes. *Customizable modeling of hierarchical planar subdivision.* Ph.D. thesis, PUC–Rio, Brazil, August 1995. (in portuguese).

[15] R. Laurini and F. Milleret-Raffort. Topological reorganization of inconsistent geographical databases: a step towards their certification. *Computer & Graphics*, 18(6):803–813, 1994.

[16] R. G. Newell and T. L. Sancha. The difference between CAD and GIS. *Computer-Aided Design*, 22(3):131–135, April 1990.

[17] K. Weiler. Topology as a framework for solid modelling. In *Proceedings of Graphics Interface '84*, pages 53–56, 1984.

[18] H. Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260, June 1984.

[19] M. J. Egenhofer, E. Clementini, and P. Di Felice. Topological relations between regions with holes. *Int. Journal of Geographical Information Systems*, 8(2):129–142, 1994.

[20] B. P. Brugeger and A. U. Frank. Hierarchies over topological data structures. In *Proceedings of ASPRS/ACSM – Annual Convention*, volume 4, pages 137–145, April 1989.

[21] E. Puppo and G. Dettori. Towards a formal model for multiresolution spatial maps. In *Advances in Spatial Databases (Proc. of SSD'95)*, volume 951 of *Lecture Notes in Computer Science*, pages 152–169. Springer, 1995.

[22] D. H. Ballard. Strip trees: a hierarchical representation for curves. *Commun. of the ACM*, 24:310–321, 1981.

[23] O. Gunther. The arc tree: an approximation scheme to represent arbitrarily curved shapes. *Computer Vision, Graphics, and Image Processing*, 51(3):313–337, September 1990.

[24] H. Raafat, Z. Yang, and D. Gauthier. Relational spatial topologies for historical geographical information. *Int. Journal of Geographical Information Systems*, 8(2):163–173, 1994.