**The Visual Computer manuscript No.**
(will be inserted by the editor)

**Pierre Kraemer · David Cazier · Dominique Bechmann**

# Extension of half-edges for the representation of multiresolution subdivision surfaces

**The original publication is available at www.springerlink.com/content/31122h5w513w8418 published in The Visual Computer, Volume 25, N. 2**

**Abstract** We address in this paper the problem of the data structures used for the representation and the manipulation of multiresolution subdivision surfaces. The classically used data structures are based on quadtrees, straightforwardly derived from the nested hierarchy of faces generated by the subdivision schemes. Nevertheless, these structures have some drawbacks: specificity to the kind of mesh (triangle or quad); the time complexity of neighborhood queries is not optimal; topological cracks are created in the mesh in the adaptive subdivision case.

We present in this paper a new topological model for encoding multiresolution subdivision surfaces. This model is an extension to the well-known half-edge data structure. It allows instant and efficient navigation at any resolution level of the mesh. Its generality allows the support of many subdivision schemes including primal and dual schemes. Moreover, subdividing the mesh adaptively does not create topological cracks in the mesh. The extension proposed here is formalized in the combinatorial maps framework. This allows us to give a very general formulation of our extension.

**Keywords** geometric modeling · topological model · combinatorial maps · multiresolution meshes · subdivision surfaces

## 1 Introduction

Modeling with multiresolution subdivision surfaces is gaining more and more popularity among the computer graphics community [37]. Indeed, this technique offers many advantages. It combines the simplicity and topological generality of subdivision surfaces, with the possibility of editing a mesh at different resolution levels, leading to small to large scale smooth modifications of the surface.

Many tools have been provided in recent years to deal with such surfaces, like inserting non-smooth features [3], Boolean operations [2], features cut-and-paste [4], trimming [5], ... However, the authors of such tools generally focus their interest on the quality of the resulting surfaces, and not on the underlying data structure.

There are different ways to obtain a multiresolution subdivision surface. One can start from a manually designed base mesh and generate the finer levels using subdivision schemes. Another way is to use remeshing techniques such as MAPS [19] that starts from an arbitrary fine mesh and produce a multiresolution subdivision mesh. Regardless of the way these surfaces are built, we focus in this paper on models and data structures that support multiresolution edition and adaptive subdivision.

Multiresolution edition means deformation of the surface at any resolution level of the mesh. An edit on some fine level will lead to a local deformation, while an edit on some coarse level will lead to a large scale – detail preserving – deformation of the surface. Each modification is repercuted on the finer levels – known as the synthesis process – and on the coarser levels – known as the analysis process.

By adaptive, we mean that the subdivision depth can differ along the mesh according to some geometrical criteria (local curvature, distance to limit surface, point of view, ...). Due to the multiresolution edition support, adaptivity has to be managed in a fully dynamic way. During each update, subdivision depth can be locally increased to fulfill the criteria or decreased if the constraint is relaxed.

In this context a general and efficient data structure should provide the following features:

- direct and simple access to all intermediate resolution levels of the mesh, as editing operations can be performed on any resolution level;

P. Kraemer, D. Cazier, D. Bechmann
LSIIT, UMR CNRS 7005
Université Louis Pasteur
Strasbourg, France
E-mail: {kraemer,cazier,bechmann}@lsiit.u-strasbg.fr

- very efficient adjacency queries on each level, as these queries are the most widely used in the synthesis and analysis processes of multiresolution edition;
- support of adaptive subdivision, i.e. subdivision at variable depth on the mesh, as it enables strong memory savings;
- support of many subdivision schemes in the same general data structure, as this can be a great advantage to stronlgy reduce the development cost of modeling tools.

### 1.1 Previous work

The classical data structure used for encoding multiresolution subdivision surfaces is the quadtree [10], which is naturally derived from the nested hierarchy of faces generated by the subdivision schemes. The only represented topological entity is the face, limited to triangles or squares. Quadtrees are not a topological structure: relations between the cells are not explicitely represented. This leads to an easy and straightforward implementation.

But this simplicity can turn into a drawback. When performing adaptive subdivision in a mesh, neighboring faces may have different subdivision depths. This causes holes or cracks in the mesh (see figure 1a). The boundary of the non-subdivided face cannot fit to its new neighborhood as it cannot be anything else than a triangle or a square. Techniques have been developped to fix this problem. These are generally based on the construction of restricted quadtrees [17,25] which does not allow two neighboring faces to have more than one resolution level of difference. Additional triangles are then added to the mesh to fix the cracks (see figure 1b). These triangles have to be removed in order to update the subdivision in this area. These additional treatments mitigate the advantages of the simplicity of the structure.

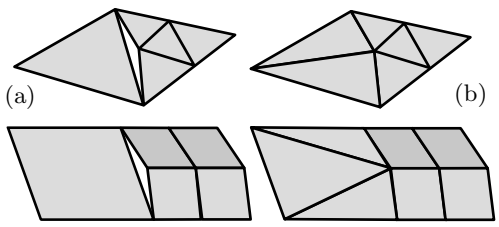



**Fig. 1** Topological cracks with quadtrees

The adjacency operators have to be specifically developed for triangle or square meshes. Moreover, the neighborhood relations have to be computed and are executed in a time linear in the number of resolution levels – i.e. the depth of the quadtree. Data structures and algorithms have been proposed to improve the time complexity of these queries: linear quadtrees [29,20] are a pointerless structure that store the leaves of a quadtree linearly in an array, and resolve neighborhoods queries using constant time arithmetic operations on indices. However, this kind of structure is not well suited for dynamic and adaptive quadtrees, as we explain later.

The Qreg data structure presented in [31] proposes a two-layer representation that exploits the regularity of meshes, notably those generated by a subdivision process. The first layer stores the adjacencies between regions of the original mesh (in the simpler case a region corresponds to a single face). The second layer associates a submesh to each of these regions that satisfies regularity constraints. These regular subdivisions can be refined hierarchically. Adaptive subdivision can be achieved by refining each region to a different level. This leads to a limited adaptivity since subdivision has to be achieved regularly within each region. Nevertheless, a fine-grain adaptivity is possible using a tree structure, but this reintroduces the drawbacks of the tree structures, and this possibility is not much developed in the paper.

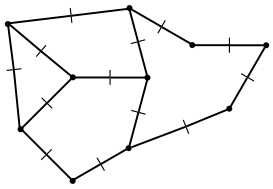


**Fig. 2** Polygonal mesh represented with half-edges

The half-edge data structure [36] is a very general and efficient topological structure: it can represent arbitrary polygonal meshes (see figure 2) and neighborhood queries are executed in constant time. This data structure has already been used in the context of subdivision surfaces [30], but only the finest resolution level is represented: each subdivision step is effectively applied on the half-edge structure using some Euler operators. After each subdivision step, the mesh describing the previous resolution level is lost. This structure has also been used in a multiresolution context, but in a decimation approach based on edge collapses on triangular meshes [26]. There is no notion of resolution levels here, and only the base mesh and the current decimated mesh are available for traversal.

### 1.2 Our contribution

We present in this paper a multiresolution extension of the half-edge data structure. This structure can efficiently be used to represent and manipulate multiresolution subdivision surfaces. Within the hierarchy, each resolution level of the mesh is instantly available as a "single resolution" half-edge structure. Thus, it inherits the advantages of this structure: polygonal meshes can

be managed and adjacency queries are executed in an optimal constant time on each resolution level.

Adaptive subdivision is handled in a simple way without creating topological cracks in the mesh. Indeed, being a cellular topological model, half-edges can manage polygonal faces. This allows to always maintain a correctly connected mesh. Moreover, the generality of this structure enables to use it with many subdivision schemes, including primal and dual schemes. Despite this generality, memory requirements are shown to be quite equivalent to that of the quadtree structures.

The half-edge data structure is an implementation of the theoretical model of the 2-dimensional combinatorial maps [15,35] (or 2-maps). This model and its extensions [21,22] have gained popularity through numerous works [13,9]. Combinatorial maps have already been used in a multiresolution context, but in the image processing field for a segmentation application using decimation algorithms [6,7,16]. In this work, combinatorial pyramids are defined. Their definition is quite close to the one we give in this work. However, the goals and the way to construct the hierarchy are different: combinatorial pyramids are constructed from a fine mesh, simplified by cell contraction and suppression operations, while we construct the hierarchy by successive refinements of a coarse mesh.

The definition of the operators meets strong constraints and is closely related to the construction of the hierarchy and to the algorithms that are applied on it. Our model needs more freedom in the definition. Indeed, we want it to be usable with many subdivision schemes that use different refinement operators (edge split, edge insertion, edge flip, ...).

2-maps can be seen as a specialization of a more general model called hypermaps. We are going to formulate our multiresolution extension in this general formulation. Then, we deduce the definition of the multiresolution 2-maps, and translate it into a data structure that is an extension of the classical half-edge data structure. This approach makes it possible to apply the proposed multiresolution extension to all the models derived from the combinatorial maps. In particular, it can be applied to higher dimensions models like 3-maps in order to represent multiresolution volumetric meshes or to generalized maps or G-maps like in [16]. We did not choose to use G-maps in this work as this would have be more memory consuming with the only advantage of being able to manage unoriented surfaces which is not crucial in our current context.

### 1.3 Paper outline

We begin this paper by presenting the classical half-edge data structure, and expressing this data structure in the combinatorial maps framework. In the third section we expose our multiresolution extension to this model, and translate it into a data structure.

Then we show how it can be efficiently applied to the representation of adaptive multiresolution subdivision surfaces. In the fifth section we compare this structure with the classical quadtrees in terms of time complexity and memory requirements.

Finally we conclude and give some future perspectives to this work.

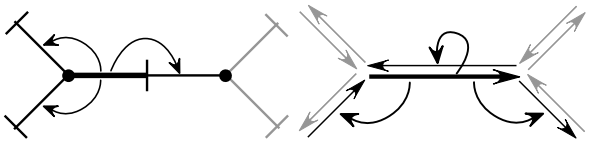
## 2 Half-edges and combinatorial maps

In this section, we first make some recalls about the half-edge data structure and the combinatorial 2-maps model which is its formalization. Then, we define the hypermaps which is the general framework for combinatorial maps.

### 2.1 Half-edges and 2-maps

A 2-dimensional mesh consists in the discretization of a surface in a cellular complex composed of cells of different dimensions (faces, edges, vertices) connected by adjacency relationships. The half-edge data structure uses the adjacencies between edges to represent the topology of the mesh of orientable 2-manifold. As illustrated above, arbitrary polygonal meshes can be represented.

Each edge of the mesh consists in two symmetric half-edges. This structure comes in two definitions: each half-edge is associated with either a vertex-edge pair (figure 3(a)), or a face-edge pair (figure 3(b)). Each half-edge is linked to its *opposite*, *next* and *previous* half-edges.

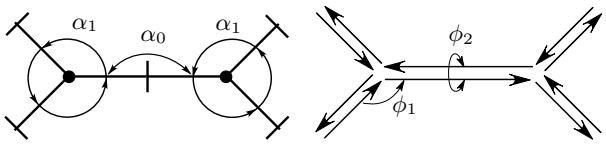


(a) Vertex-edge pair (b) Face-edge pair

**Fig. 3** The half-edge data structure

Two types of information are needed to fully describe an object: the *topological relationships*, and the *embedding*, i.e. the geometrical data associated to each topological entity. In most of the cases, we simply associate 3D points to the vertices of the mesh. The embedding of the other cells is computed by linear interpolation. More evolved embedding models can also be used, attaching for example curves to the edges or surface patches to the faces. The half-edges attached to a same topological entity share links to the corresponding embedding information.

When implementing such a structure, a mesh can be simply represented by a set of half-edges. The topological links are materialized by pointers between the half-edges. The *previous* pointer may not be stored as it is




(a) Primal 2-map (b) Dual 2-map

**Fig. 4** The 2-maps

not necessary if the operators that deal with the *next* link guarantee to never break the cycle of half-edges. It is the same choice as implementing simply or doubly linked circular lists (the pointer can be stored if the storage cost is not crucial). In case of vertices-only embedding, all the half-edges attached to the same vertex share a pointer to the same 3D point. A simple example of a C++ implementation of the half-edge data structure is shown in the appendix (listing A.1).

The model of the 2-maps is the formalization of the half-edge data structure. A 2-map is a 3-tuple:

$$M = (D, \alpha_0, \alpha_1)$$

where $D$ is a finite set of darts (the *half-edges*); $\alpha_0$ (the *opposite* pointer) is an involution on $D$ without fixed point, i.e. a permutation such that $\alpha_0(\alpha_0(x)) = x$ and $\alpha_0(x) \neq x$, for all $x$ in $D$; and $\alpha_1$ (the *next* pointer) is a permutation on $D$. The *previous* pointer of the half-edge data structure is here implicitly represented by the inverse permutation $\alpha_1^{-1}$.

The 2-maps also come in two definitions, called primal and dual, corresponding respectively to the vertex-pair and face-pair versions of the half-edge data structure. In both cases the darts of the map are linked by one involution and one permutation. To differentiate the two versions, different names are given to these relations. In the primal case (figure 4(a)), the involution is called $\alpha_0$ and the permutation, which turns around vertices is called $\alpha_1$. In the dual case (figure 4(b)), the involution is called $\phi_2$ and the permutation, which turns around faces is called $\phi_1$. One can easily switch from one version to the other as we have $\phi_2 = \alpha_0$ and $\phi_1 = \alpha_0 \circ \alpha_1$.

The embedding of a 2-map is a function which associates a geometrical entity to a topological cell. For 0-embedding for example, the function $em_0$ associates a 3D point to each vertex of the map, i.e. all the darts attached to a same vertex have the same image by the function $em_0$.

### 2.2 The hypermaps

Hypermaps are the general framework from which all the models derived from the combinatorial maps can be obtained by adding some constraints. A hypermap is a $(n+1)$-tuple:

$$G = (D, \alpha_0, \ldots, \alpha_{n-1})$$

where $D$ is a finite set of darts, and the $\alpha_i$, with $i \in [0, n-1]$, are permutations on $D$.

For example, the 2-maps described above can be recovered from this definition by fixing $n = 2$ and constraining the $\alpha_0$ permutation to an involution.

## 3 The multiresolution extension

In this section we expose our multiresolution extension of the combinatorial maps. We define it firstly on the hypermaps, then by adding some constraints we recover the definition of the multiresolution 2-maps.

Finally, we translate this model into a data structure which can be seen as an extension of the half-edge data structure.

### 3.1 Multiresolution hypermaps

A multiresolution hypermap is defined as a hierarchy of hypermaps. It is composed of a set of darts $D$ and topological relations between these elements.

As shown in figure 5, the darts of $D$ are differentiated by the resolution level in which they are introduced in the map. We can see here an example of three consecutive resolution levels. The darts of the starting map (or level 0 map) belong to the set $L^0$ (also named $D^0$). The set of new darts $L^1$ is introduced in the map on resolution level 1 and added to $L^0$ to form the set $D^1$. On the next resolution level, the set of darts $L^2$ is added to $D^1$ to form the set $D^2$, and so on...

More formally, in a multiresolution map, the sets of darts form a sequence $\{D^i\}_{i\geq 0}$ of sets such that:

$$D^0 \subset D^1 \subset D^2 \subset \ldots \subset D^i \subset \ldots$$

The set of darts $D^i$ contains all the darts introduced in the resolution levels inferior or equal to $i$. The total number of darts is equal to the number of darts needed to describe the finest resolution level. We have the following formulations:

$$D^i = \bigcup_{j=0}^{i} L^j \qquad D = \bigcup_{i\geq 0} L^i$$

The topological relations between the darts are defined on every resolution level. We choose to note it by indexing the relation with the queried level: for any permutation $\sigma$ on $D$ and for any dart $d \in D^i$, $\sigma^i(d)$ is the dart linked to $d$ by the permutation $\sigma$ on resolution level $i$.

Thus, a multiresolution hypermap is defined as a $(n+1)$-tuple:

$$G = \left(D, \{\alpha_0^i\}_{i\geq 0}, \ldots, \{\alpha_{n-1}^i\}_{i\geq 0}\right)$$

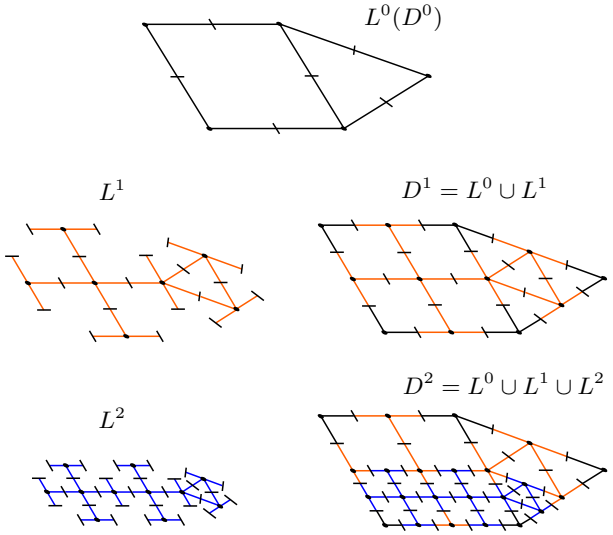



**Fig. 5** Successive sets of darts

such that for all $i \geq 0$, the $(n+1)$-tuple:

$$G^i = (D^i, \alpha_0^i, \ldots, \alpha_{n-1}^i)$$

is the hypermap describing the resolution level $i$.

The embedding function is also defined on every resolution level. For 0-embedding, for any dart $d \in D^i$, $em_0^i(d)$ is the 3D point associated with the vertex of $d$ on resolution level $i$.

### 3.2 Multiresolution 2-maps

Multiresolution 2-maps are obtained from the above general definition by fixing $n = 2$, and constraining the $\alpha_0^i$ relations to involutions.

A multiresolution 2-map is defined as a 3-tuple:

$$M = \left(D, \{\alpha_0^i\}_{i\geq 0}, \{\alpha_1^i\}_{i\geq 0}\right)$$

such that for all $i \geq 0$, the 3-tuple:

$$M^i = (D^i, \alpha_0^i, \alpha_1^i)$$

is the 2-map describing the resolution level $i$.

Let us illustrate the topological relations in a multiresolution 2-map. Figure 6(a) shows an edge at resolution levels $l$ and $l+1$. On level $l$ the edge is formed by two darts $d_1$ and $d_2$ linked by $\alpha_0^l$. On level $l+1$, a new vertex is inserted, introducing two new darts $d_3$ and $d_4$. These darts are linked to $d_1$ and $d_2$ by $\alpha_0^{l+1}$. The relations of level $l$ between the darts of $D^l$ are not lost by this operation.

Figure 6(b) shows a vertex at resolution levels $l$ and $l+1$. On level $l$ the vertex is composed of two darts $d_1$ and $d_2$, linked by the permutation $\alpha_1^l$. On level $l+1$, a new dart $d_3$ is introduced in the vertex and thus in the

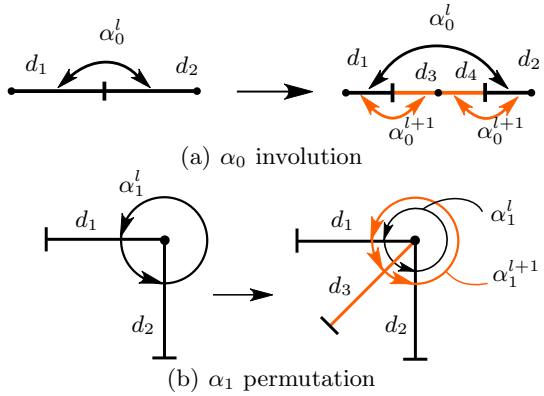



**Fig. 6** Toplogical relations in a primal multiresolution 2-map

permutation $\alpha_1^{l+1}$ . Here again, the relations of level $l$ are still available.

The dual definition of the multiresolution 2-maps is simply obtained by combining the relations of the primal definition. We define $\phi_1^i = \alpha_0^i \circ \alpha_1^i$, and $\phi_2^i = \alpha_0^i$. We illustrate now the topological relations in a dual multiresolution 2-map.

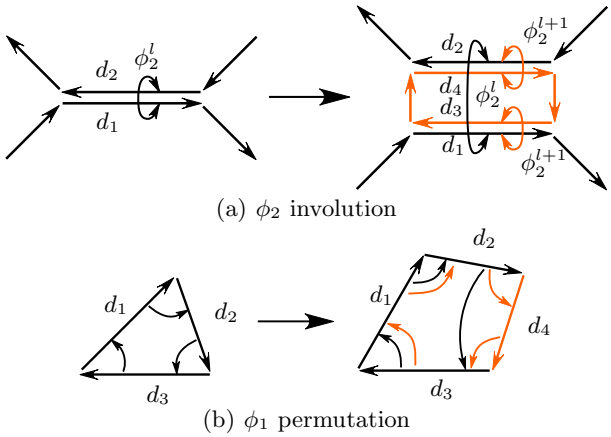



**Fig. 7** Topological relations in a dual multiresolution 2-map

Figure 7(a) shows an edge at resolution levels $l$ and $l+1$. On level $l$ two faces are sewn along an edge formed by the two oriented darts $d_1$ and $d_2$ linked by $\phi_2^l$. On level $l+1$, a new face is inserted between the two oriented darts, introducing some new darts $d_3$ and $d_4$. These darts are linked to $d_1$ and $d_2$ by $\phi_2^{l+1}$, without altering the link between $d_1$ and $d_2$ on level $l$ by the involution $\phi_2^l$.

Figure 7(b) shows a face at resolution levels $l$ and $l+1$. On level $l$ the face is composed of three darts $d_1$, $d_2$ and $d_3$, linked by the permutation $\phi_1^l$. On level $l+1$, a new dart $d_4$ is introduced in the permutation $\phi_1^{l+1}$. Here again, the relations of level $l$ are still available.

### 3.3 Topological cells and resolution levels

We go back here to the definition of the topological cells in the 2-maps model, and show how it extends to the multiresolution definition.

In a classical 2-map, the cells of the subdivision are implicitly represented by subsets of the set of darts $D$, those subsets being formally defined as orbits. For any permutation $\sigma$ on $D$, the orbit $\langle\sigma\rangle(d)$ is the set of darts $\{d, \sigma(d), ..., \sigma^j(d)\}$, where $j$ is the smallest positive integer so as $\sigma^{j+1}(d) = d$. In other words, applied on a dart $d$ of $D$, an orbit represents the set of darts reachable from $d$ by successive applications of the permutation $\sigma$. For example, in a dual 2-map, all the darts of a face are traversed by applying successively the $\phi_1$ permutation (the *next* pointer of the half-edge structure).

Formally, in primal 2-maps the vertices are defined by the orbit $\langle\alpha_1\rangle$, the edges by the orbit $\langle\alpha_0\rangle$, and the faces by the orbit $\langle\alpha_0 \circ \alpha_1\rangle$.

In dual 2-maps the vertices are defined by the orbit $\langle\phi_2 \circ \phi_1\rangle$, the edges by the orbit $\langle\phi_2\rangle$, and the faces by the orbit $\langle\phi_1\rangle$.

In a multiresolution 2-map, the cells of the subdivision are defined on each resolution level $l$. These cells are implicitly represented by subsets of $D^l$, those subsets also being defined as orbits. These orbits are defined here exactly in the same way as in classical 2-maps, using the topological relation corresponding to the queried resolution level. For example, edges of resolution level $l$ in primal multiresolution 2-maps are defined by the orbit $\langle\alpha_0^l\rangle$.

### 3.4 Multiresolution half-edges

We now translate the model of multiresolution 2-maps into a data structure. We use here again the terminology used for the half-edge data structure.

As we have seen in the definition of the model, a set of half-edges $L^i$ is introduced on each resolution level $i$. The multiresolution half-edges can thus be stored in an array of lists, where the $i^{th}$ cell of the array contains the list containing the set $L^i$.

Thus, traversing the half-edges describing the mesh of resolution level $i$ (i.e. the darts of $D^i$) is performed by traversing the lists of darts contained in the cells of the array having an index inferior or equal to $i$.

The relations between multiresolution half-edges are materialized by pointers, indexed by the resolution level. Thus, each multiresolution half-edge stores, for each relation, an array of pointers corresponding to the links at different resolution levels. As a multiresolution half-edge does not have relations at a level which is inferior to its introduction level, this array stores only the necessary relations. Let $a$ be the introduction level of a half-edge $h$, and $k$ be the maximum resolution level of the mesh: $h$ stores only $(k - a)$ links for each relation. Thus, the pointer to the linked half-edge on resolution level $i$ is stored in the $(i - a)$th cell of the array.

Each multiresolution half-edge stores such an array for the *opposite* and *next* relations (the *previous* link being unnecessary if a consistent cycle is always maintained). Another similar array is stored for the pointers to the embedding, so that the vertices of the mesh can have a different embedding on each resolution level. The introduction level of the half-edge is also stored (one byte is far enough for this).

By fixing the *level* parameter to a given value $l$ and considering the half-edges of $D^l$, we simply traverse the mesh describing the corresponding resolution level exactly as if we were traversing a classical half-edge structure.

An example of a C++ implementation of the multiresolution half-edge data structure is shown in the appendix (listing A.2).

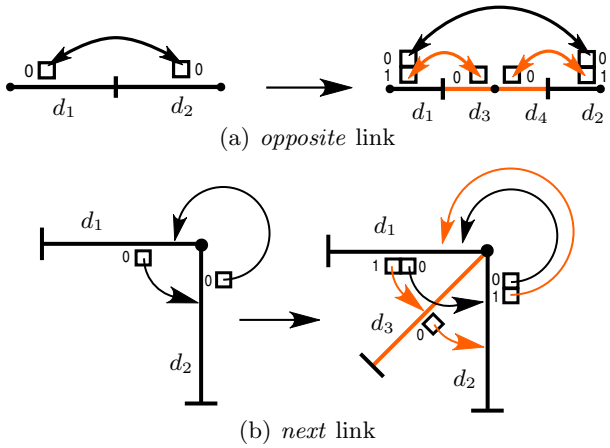
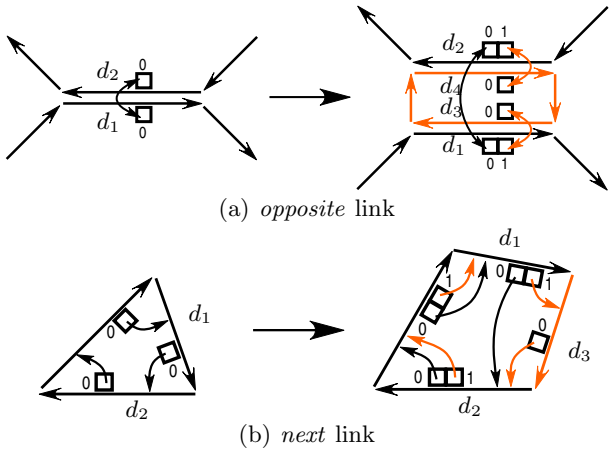



**Fig. 8** Indexation of the links in primal multiresolution half-edges

Figures 8 and 9 illustrate the indexation of the relations in the same way as before, but now showing the arrays of pointers corresponding to the links on resolution levels 0 and 1.

We see in figure 8(a) and 9(a) that the array of *opposite* pointers of $d_1$ and $d_2$ contains two elements on level 1. For these two half-edges whose introduction level is 0, the cell of index 1 contains the pointers of resolution level 1. For the half-edges $d_3$ and $d_4$ whose introduction level is 1, it is the cell of index 0 that contains the pointers of resolution level 1.

In figures 8(b) and 9(b), one can notice that for some of the half-edges, the *next* link is the same on the two levels. We will see later how this redundant storage can be avoided.




(a) *opposite* link

(b) *next* link

**Fig. 9** Indexation of the links in dual multiresolution half-edges

## 4 Application to multiresolution subdivision surfaces

In this section, we will see how the data structure that we just defined can efficiently represent multiresolution subdivision surfaces. First, we recall some notions about subdivision surfaces and their multiresolution extension.

### 4.1 Background notions

*Subdivision surfaces* The basic idea of subdivision surfaces is to define a smooth surface as the limit of an infinite sequence of successively refined polyhedral meshes. Each finer mesh is obtained from a coarse one by using a set of refinement rules which defines a *subdivision scheme*. One must distinguish clearly here the two steps of the refinement process. In the first step, the topology of the mesh is subdivided (for example by inserting vertices). During the second step, the geometry is computed, assigning a 3D point to each vertex of the finer mesh. Different schemes lead to limit surfaces with different smoothness characteristics.

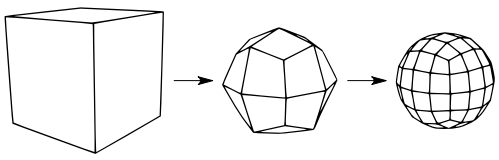
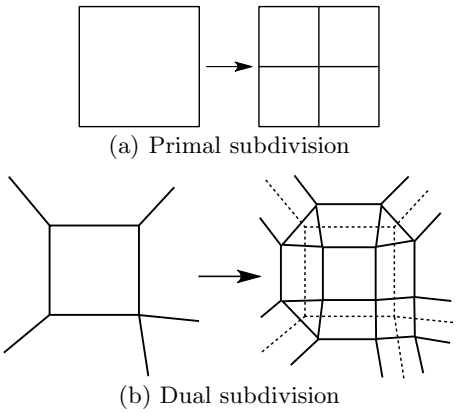


**Fig. 10** 2 subdivision steps on a cube

The topology refinement process is usually classified in two families called primal and dual subdivision. In the *primal* subdivision schemes, the *faces* of the mesh are split into several faces by cutting their boundary edges and connecting the created vertices by new edges to form new faces. The Loop scheme [23], the Catmull-Clark scheme [8], and the Butterfly scheme [14,39] are well-known primal schemes. An example of primal subdivision is given in figure 11a, where one square face is subdivided into four faces.

In the *dual* subdivision schemes, the *vertices* of the mesh are split, the faces of the coarse mesh are contracted and the created holes are filled with new faces. The Doo-Sabin scheme [12,11] and the Midedge scheme [27] are well-known dual schemes. An example of dual subdivision is given in figure 11b.



(a) Primal subdivision

(b) Dual subdivision

**Fig. 11** Topological subdivision

The geometry computing process is also usually classified in two families called interpolating and approximating subdivision. These two families share the way the new positions are computed: for a vertex of the finer mesh, its new position is computed *locally* by applying a mask on the positions of the neighboring vertices in the coarse mesh. The size and the coefficients of the mask differ from one scheme to the other, leading to different kinds of limit surfaces.

In an approximating scheme, the positions of all the vertices of the finer mesh are computed and moved closer to the limit surface. The Loop scheme, the Catmull-Clark scheme and the Doo-Sabin scheme cited above are approximating. In an interpolating subdivision scheme, the vertices of the coarse mesh already lie on the limit surface and only the positions of the new vertices of the finer mesh are computed. The Butterfly subdivision scheme cited above is interpolating.

For additional details, we refer the reader to this course on subdivision surfaces [38].

*Multiresolution* Multiresolution subdivision surfaces extend the concept of subdivision surfaces by keeping all the intermediate meshes between the coarser one and the finer one (which will be called levels of resolution), and by allowing detail vectors to be introduced at each of

these levels. The detail vectors are computed as differences between levels and are expressed in a local frame. Hence, a finer mesh is computed by adding detail vectors to the subdivided coarse mesh. This representation was introduced among others in [24,40].

A close connection exists between multiresolution surfaces and wavelets, and in particular two operations known as analysis and synthesis. Analysis computes positions of vertices on a coarse level $i-1$ by applying a reverse subdivision filter [1] or a non-shrinking smoothing filter [33] to the points on level $i$. Detail vectors on level $i$ are computed as differences between the two levels. Conversely, synthesis reconstructs the data on level $i$ by subdividing the mesh of level $i-1$ and adding the detail vectors.

A multiresolution subdivision surface can be constructed either starting from a level 0 mesh and creating the finer levels by applying a subdivision scheme (also called "coarse to fine" approach), or starting from a fine mesh and constructing the coarser levels by applying the analysis process (also called "fine to coarse" approach). In the latter case, the starting mesh must have subdivision connectivity, i.e. it must have the same connectivity as if it had been generated by a subdivision surface scheme. The problems of subdivision connectivity and of processing arbitrary meshes are widely studied (for example in [34,19]).

Once a multiresolution subdivision surface is constructed, the mesh can be edited at any level of resolution. An edit at some coarse level will lead to a large scale deformation, keeping the high frequency informations thanks to the detail vectors. An edit at some finer level will lead to a more localized deformation. At each edition, the finer levels are updated by the synthesis process, and the coarser levels are updated by the analysis process, computing the detail vectors at the same time.

### 4.2 Primal schemes

During a step of a primal subdivision scheme, edges of the mesh are cut, and new edges are introduced to form new faces. We use the *primal* version of the multiresolution half-edge structure to handle this kind of subdivision. The operations are executed on the mesh in the same way as in a standard half-edge structure, except that the new half-edges are introduced and linked with the others in the new resolution level – thus keeping available the old resolution level.

Figure 12 illustrates a detail of a triangle mesh at two consecutive resolution levels $l$ and $l+1$ using the Loop subdivision scheme. The half-edges in bold orange are those that were introduced in level $l+1$.

Figure 13 illustrates a detail of a square mesh at two consecutive resolution levels using the Catmull-Clark subdivision scheme.

We can notice that in primal subdivision schemes, the valence of the existing vertices is not modified by a subdivision step. This means that for a multiresolution half-edge $h$ introduced on a resolution level $l$, its *next* link do not change in the finer resolution levels. Thus, it is unnecessary to redefine on each finer resolution level that the *next* link of $h$ is the same than on resolution level $l$. In this particular case, we can store a simple pointer instead of the array of *next* pointers, and use this pointer on every resolution level finer than $l$. In a more general case, it can be solved in the same way as adaptivity is managed, as we explain in the following.

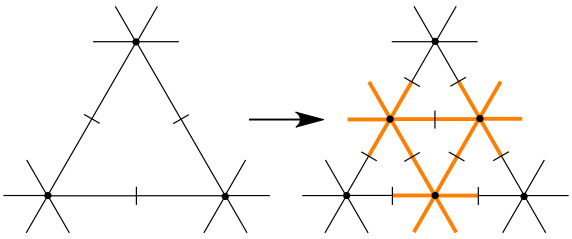


**Fig. 12** Loop with multiresolution half-edges

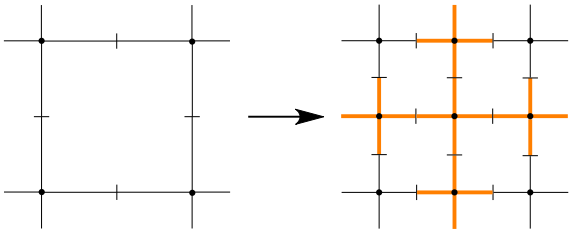


**Fig. 13** Catmull-Clark with multiresolution half-edges

### 4.3 Adaptivity

In an adaptively subdivided mesh, the subdivision steps are not always performed regularly all over the mesh. Different areas of the mesh reach different subdivision depths. The goal is here to save the memory that would have been spent performing a regular subdivision all over the mesh. The same precision in the approximation of the surface is not needed in the flat areas, where large faces are a sufficiently good approximation, and in the highly detailed or high curvature areas, where smaller faces are needed. Different metrics or criteria can be used to decide when to stop or continue the subdivision.

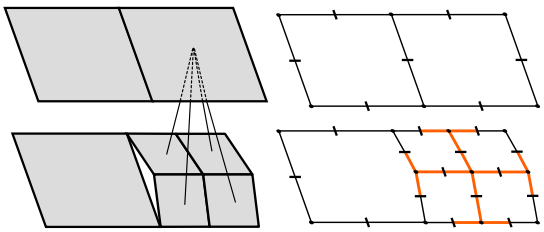
Following a given criterion, it can be decided that a face is no longer subdivided starting from a given level. This implies that some edges of the mesh are no longer cut between two consecutive levels. This means that for a multiresolution half-edge $h$ whose corresponding edge stops being subdivided starting from a resolution level $l$, its *opposite* link does not change in the finer resolution levels. Thus, it is unnecessary to redefine this link on each finer resolution level.

This is managed in the following way: if a relation does not change for a multiresolution half-edge between two resolution levels, then we simply do not redefine anything on the finer resolution level (i.e. its corresponding array of pointers does not grow and no pointer is duplicated). In consequence, the way of accessing the topological links has to be changed, as shown in listing A.3: if nothing is defined on the queried resolution level (i.e. if the array of pointers is too small for the queried resolution level), we pick the information corresponding to the maximum resolution level where something was defined (i.e. the last pointer in the array). The same treatment is also applied to access the embedding information.

This can sound limiting, as it forbids a modification of a relation at a resolution level $l$, if no modification was applied since some previous resolution level (there cannot be "holes" in the array). But this suits well to the context of subdivision surfaces. Indeed, a face is not subdivided on level $l$ if it has not already been subdivided on the previous resolution levels. Nevertheless, if it is necessary in some application, this can be solved by duplicating the last pointer of the array until the cell corresponding to the level where a modification happens (i.e. filling the "holes" in the array), or by replacing the array of pointers by a lookup table (but in this case, access to the pointer of a given resolution level would not be executed in constant time).

As already discussed above, in the structures based on quadtrees, adaptivity generates topological cracks in the mesh, at the meeting of areas of different resolution levels. Holes appear in the mesh between neighboring faces of different refinement depths (see figure 14a). This happens because the only topological entity that is manipulated here is the face, while the subdivision process also involves the edges of the mesh.

In a multiresolution map, the subdivision of a face implies the subdivision of its edges. The neighboring faces are affected by this operation even if they are not subdivided themselves. Indeed, as it can be seen in figure 14b, the left face is now a pentagonal face. Multiresolution maps being a cellular model, this face is here naturally managed.



(a) Quadtree (b) Multiresolution half-edges

**Fig. 14** Adaptive Catmull-Clark subdivision with a quadtree and with multiresolution half-edges

Figure 15 illustrates an adaptively triangle mesh at two consecutive resolution levels. Between these levels, only the central face is subdivided. Thus only three edges are cut and the three surrounding faces simply become here quadrilateral faces.

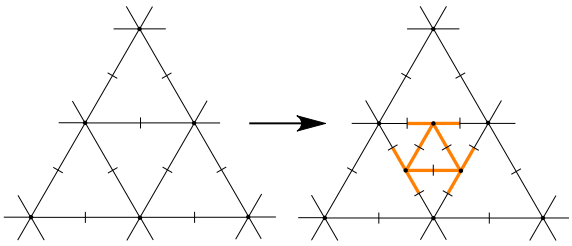


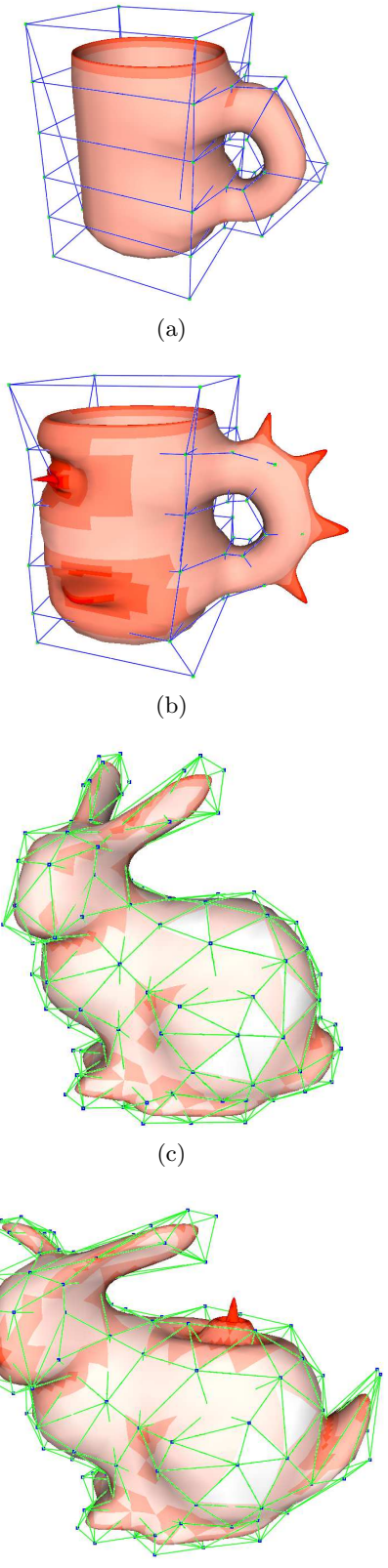
**Fig. 15** Adaptive Loop subdivision

Figure 16 shows examples of objects modeled with multiresolution half-edges and obtained by adaptive Catmull-Clark (16(a) and 16(b)) or Loop (16(c) and 16(d)) subdivision. The color is function of the resolution depth of the faces. The adaptivity is here automatically driven by a geometric criterion. Subdivision depth follows the dihedral angle between adjacent faces: a face is subdivided only if the angle between its normal and the normals of its neighboring faces exceeds a given threshold. Different criteria can be used depending on the goals of the applications. The resulting mesh is here automatically obtained during the edition by applying subdivision or coarsening until satisfaction of the geometric criterion in the area concerned by the edition.

A restriction is also introduced to forbid adjacent faces to have more than one level of difference. Although this restriction leads to more subdivision than needed around the high curvature areas, it allows to have always full subdivision masks. This means that all the neighboring vertices needed to compute a position during the subdivision process are always present in the mesh. This avoids the temporary computation of the eventually missing vertices. The resulting precision in the approximation of the limit surfaces is also better as the size of the faces decrease progressively around these areas. However, as this restriction leads to more subdivision than needed and as it is not necessary in the definition of the model, it can be disabled if the memory usage is a crucial parameter in an application.

### 4.4 Dual schemes

During a step of a dual subdivision step, the vertices are split and new faces are introduced in the mesh. We use here the *dual* version of the structure to handle this kind of subdivision. Here again, the operations on the mesh are done in the same way as in a standard half-edge



(a) (b) (c) (d)

**Fig. 16** Objects modeled with multiresolution half-edges (Catmull-Clark and Loop schemes)

structure, except that the new half-edges are introduced and linked in the new resolution level.

In dual subdivision schemes it is the number of edges of the faces that is not modified during a subdivision step. Here again, once the *next* link is set on the introduction level of a half-edge, it does not change in the finer levels. The storage of this redundant information can be avoided in the same way as explained above.

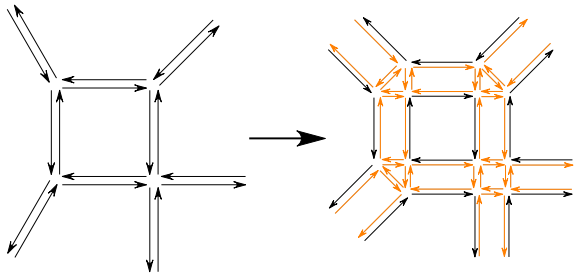


**Fig. 17** Doo-Sabin with multiresolution half-edges

Figure 17 illustrates a detail of a mesh at two consecutive resolution levels using the Doo-Sabin subdivision scheme.

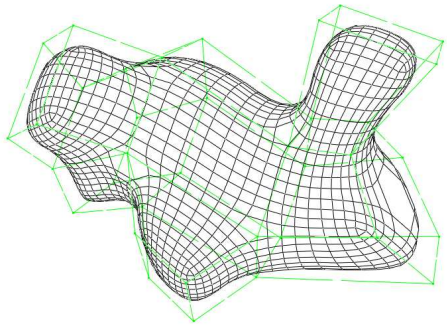


**Fig. 18** Object modeled with multiresolution half-edges (Doo-Sabin scheme)

Figure 18 shows an example of object modeled with multiresolution half-edges and obtained by Doo-Sabin subdivision.

## 5 Comparison

As the quadtree is a widely used data structure for the representation of multiresolution adaptive subdivision surfaces, let us compare to it in terms of time complexity and memory space requirements.

### 5.1 Time complexity

In the framework of multiresolution subdivision surfaces, the most frequently executed operations are adjacency queries. These are needed as well while applying the subdivision scheme, as while executing the analysis algorithm.

In a multiresolution half-edge structure, basic adjacency queries are executed in constant time, whatever the considered resolution level. Indeed, adjacent cells are retrieved directly by following pointers indirections. More evolved neighborhood algorithms are then built using these constant time operators. The time complexity of these algorithms is typically linear to the size of the considered neighborhood.

Let us enumerate some of the basic constant time adjacency operators. Vertex-vertex adjacency (figure 19a) is resolved following 1 pointer that leads from one half-edge of a given vertex to one half-edge of a neighboring vertex; edge-edge adjacency (figure 19b) is resolved following 1 or 2 pointers and face-face adjacency (figure 19c) following 1 pointer. These operators are executed with the same efficiency on any resolution level (as every level can be traversed exactly as a standard half-edge structure).

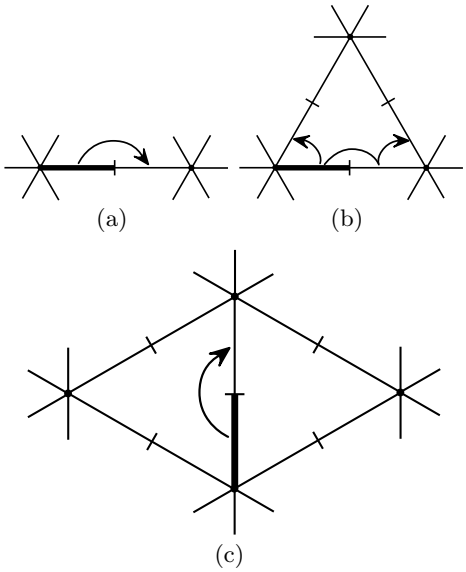



**Fig. 19**

Given these operators, more evolved algorithms can be built. Figure 20 illustrates a widely used algorithm that consists in visiting all the neighbors of a given vertex. The geometry of the central vertex on level $l+1$ is computed as a combination of the positions of its neighbors on level $l$.

In the primal version of the structure, starting from the vertex of the half-edge $h$, traversing all its neighboring vertices on level $l$ is performed as follows: the vertex orbit of $h$ is traversed following the *next* pointers, and for each half-edge of this orbit, the adjacent vertex is retrieved by crossing the corresponding edge following the *opposite* pointer of the considered level ("jumping" directly over the higher resolution mesh). This algorithm is executed in an optimal time linear in the number of neighbors of the vertex, and does not depend on the resolution level or on the maximum resolution depth of the mesh.

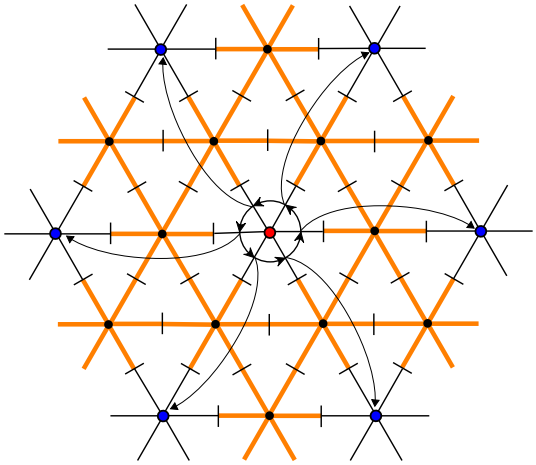


**Fig. 20** Neighbors traversal

Listing A.4 in the appendix shows an example of how this kind of algorithm can be implemented in the data structure.

Similar algorithms can be written for example to traverse all the vertices of a face, or all the adjacent faces of a face on level $l$, whatever the resolution level of these neighboring faces – in the adaptive setting they can have a resolution level inferior or equal to $l$.

In a classical forest of quadtrees, neighborhood queries within a single quadtree are resolved in the standard way by ascending the tree to the least common parent when attempting to find the neighbor across a given edge. Neighborhood relations between adjacent trees are resolved explicitly at the level of a collection of roots, i.e. faces of the level 0 mesh. These adjacency queries are performed in a time linear to the the depth of the tree, i.e. the maximum level of the multiresolution structure. Even if in practice this execution time is bounded by a small value, this value is not 1. These adjacency queries being the most used operators when editing a multiresolution surface, this improvement is not negligible.

We have seen that constant time adjacency queries algorithms have been developped for linear quadtrees, as well for square [29] as for triangle [20] meshes. But besides this, linear quadtrees have numerous drawbacks.

One of the strongest ones is that the level 0 mesh – i.e. the roots of the quadtrees – has to be fixed and known *a priori* in order to be able to resolve adjacencies

between faces that do not belong to the same base face – i.e. the same quadtree. These inter-quadtrees queries have only been defined for a few kind of level 0 meshes: icosahedron, octahedron or tetrahedron.

Moreover, linear quadtrees are not at all designed for adaptive subdivision: the only constant time neighborhood queries are those that find equal-sized neighbors – i.e. faces that lie on the same level of resolution – which is not the general case in the adaptive setting. The size of the indexes used for the adjacency queries resolution grows of two bits with each subdivision step. Therefore the size of the array grows in parallel, which leads automatically to a regular subdivision.

### 5.2 Storage requirements

In this section, we give an estimation of the memory cost of multiresolution half-edges and compare it with the triangular quadtree structure in the case of regular subdivision – as no general formulation can be made in the case of adaptive subdivision, and as regular subdivision is the worst case for memory requirements.

Let $|D|$ be the number of half-edges in a multiresolution half-edge structure. $|D|$ is equal to the number of half-edges that describe the mesh of maximum level. Let $h_0$ be the number of half-edges of the level 0 mesh, and $m$ the maximum level of the multiresolution structure. As in the primal subdivision schemes presented above the number of half-edges is multiplied by a factor 4 at each subdivision step, we have:

$$|D| = h_0 \cdot 4^m \tag{1}$$

To compute the total cost of the topological information in a multiresolution half-edge structure, we have to count the number of pointers stored by all the half-edges.

For the *opposite* relation, this can be done by summing the size of the arrays of *opposite* links contained in each dart. The size of this array is function of the level of introduction of the half-edge. We can notice that 3/4th of the half-edges has just been introduced at the maximum level and thus have only one *opposite* link; 3/4th of the rest, i.e. 3/16th, have two *opposite* links,... Formally, for $l$ between 1 and $m$, there are $|D| \cdot \frac{3}{4^l}$ half-edges that have $l$ elements in their array. The half-edges of the level 0 mesh have $m+1$ elements in their array.

The total number of elements of the arrays of *opposite* links of all the half-edges is then, after replacing $|D|$ using equation (1):

$$h_0 \cdot (m+1) + h_0 \cdot 3 \cdot \sum_{l=1}^{m} l \cdot 4^{m-l}$$

For the *next* relation, things are much more simpler as there is just one *next* link per half-edge, in other words $|D|$ or $h_0 \cdot 4^m$ pointers are stored.

The geometrical information is attached to the darts by an array of pointers to 3D points. If an approximating scheme is used, as the embedding is indexed by the level, this array has exactly the same size than the array of *opposite* links.

The total number of pointers stored by the half-edges is then:

$$h_0 \cdot 4^m + 2 \cdot \left( h_0 \cdot (m+1) + h_0 \cdot 3 \cdot \sum_{l=1}^{m} l \cdot 4^{m-l} \right) \tag{2}$$

The sum in equation (2) can be identified to the power series: $\sum_{n \geq 0} n \cdot x^n = \frac{x}{(1-x)^2}$, which is defined for $| x | < 1$. The sum can then be expressed like this:

$$\sum_{l=1}^{m} l \cdot 4^{m-l} \simeq 4^m \cdot \frac{\frac{1}{4}}{(1-\frac{1}{4})^2}$$

Equation (2) simplifies in:

$$\frac{33}{9} \cdot h_0 \cdot 4^m \tag{3}$$

The quadtree structures store five pointers per node for the topological information: four to the children and one to the parent. The roots of the quadtrees store seven pointers: four to the children, and three for the adjacency relations between the roots. For the geometrical information, three supplementary pointers to 3D points are stored in each node. Let $f_0$ be the number of faces of the level 0 mesh, and $m$ the maximum level of the multiresolution structure. As the number of faces is multiplied by a factor 4 at each subdivision step, the total number of pointers stored is:

$$10 * f_0 + 8 * f_0 * \sum_{l=1}^{m} 4^l \tag{4}$$

The sum in equation (4) can be identified to the power series: $\sum_{n \geq 0} x^n = \frac{1}{1-x}$, which is defined for $| x | < 1$. The sum can then be expressed like this:

$$\sum_{l=1}^{m} 4^l \simeq 4^m \cdot \frac{1}{1-\frac{1}{4}}$$

Equation (4) simplifies in:

$$\frac{32}{3} \cdot f_0 \cdot 4^m \tag{5}$$

We can now compute the ratio between the memory cost of the multiresolution half-edge structure (3) and the memory cost of the quadtree structure (5). In triangular meshes, there are three half-edges per face. We have then $f_0 = \frac{h_0}{3}$. With this we can compute the ratio:

$$\frac{\frac{33}{9} \cdot h_0 \cdot 4^m}{\frac{32}{9} \cdot h_0 \cdot 4^m} = \frac{33}{32} \tag{6}$$

We see here that our model needs only about 3% more memory space than the quadtree structures. If we take into account the storage cost of the 3D points, which

is the same for both structures, this ratio is even smaller in practice.

Using an interpolating scheme to generate the levels of resolution even make this ratio more advantageous for multiresolution half-edges than for quadtrees. Indeed, as introduced vertices already lie on the limit surface, the embedding information is no more indexed by the level. The array of pointers to 3D points stored in each half-edge can then be replaced by a simple pointer. With this, the ratio between the memory cost of the multiresolution half-edge structure and the memory cost of the quadtree structure is $\frac{30}{32}$. In this case, our model uses about 6% less memory space.

## 6 Conclusion

In this paper we have presented a new data structure for the representation of adaptive multiresolution subdivision surfaces. This new structure is defined as an extension of the well known half-edge data structure. The half-edge structure being the practical implementation of the 2-dimensional combinatorial maps, we have expressed our extension in the general theoretical framework of hypermaps, thus opening the proposed extension to all the other models derived from the combinatorial maps. In particular we plan to study the representation of multiresolution volumetric meshes by multiresolution 3-maps.

The multiresolution half-edge structure allows instant and efficient navigation at any resolution level of the mesh. Indeed, each level can be traversed just like a classical half-edge structure, inheriting its efficiency for the topological queries. Its generality allows the support of many subdivision schemes including primal and dual schemes. This generality is not obtained at the expense of storage requirements as it needs almost the same memory space as quadtree structures. Moreover, the ability to represent arbitrary faces is a great advantage when subdividing the mesh adaptively: the coexistence of different resolution levels produces non-regular faces which are here naturally supported and thus avoids the creation of topological cracks in the mesh.

We now plan to apply this model to the representation of multiresolution meshes obtained by other subdivision schemes such as hybrid quad/triangle schemes [32, 28], or the $\sqrt{3}$ scheme [18]. These schemes are not well supported by classical structures and should be easily supported by multiresolution half-edges.

## A Implementation examples

**Listing A.1** Half-edge implentation

```
class Mesh
{
  list<HalfEdge> he;
}

class HalfEdge
{
  HalfEdge * opposite;
  HalfEdge * next;

  Point3D * emb;
}
```

**Listing A.2** Multiresolution half-edge implementation

```
class MRMesh
{
  vector<list<MRHalfEdge> > levels;
}

class MRHalfEdge
{
  char intro_level;

  vector<MRHalfEdge*> opposite;
  vector<MRHalfEdge*> next;

  vector<Point3D*> emb;

  MRHalfEdge* getOpposite(char level)
  {
    return opposite[level - intro_level];
  }

  MRHalfEdge* getNext(char level)
  {
    return next[level - intro_level];
  }

  MRHalfEdge* getEmbedding(char level)
  {
    return emb[level - intro_level];
  }
}
```

## References


1. Bertram, M.: Biorthogonal loop-subdivision wavelets. Computing **72**(1-2), 29–39 (2004)
2. Biermann, H., Kristjansson, D., Zorin, D.: Approximate boolean operations on free-form solids. In: Proceedings of SIGGRAPH'01, pp. 185–194 (2001)
3. Biermann, H., Levin, A., Zorin, D.: Piecewise smooth subdivision surfaces with normal control. In: Proceedings of SIGGRAPH'00, pp. 113–120 (2000)
4. Biermann, H., Martin, I., Bernardini, F., Zorin, D.: Cut-and-paste editing of multiresolution surfaces. In: Proceedings of SIGGRAPH '02, pp. 312–321 (2002)
5. Biermann, H., Martin, I., Zorin, D., Bernardini, F.: Sharp features on multiresolution subdivision surfaces. Graphical Models **64**(2), 61–77 (2002)
6. Brun, L., Kropatsch, W.: Combinatorial pyramids. In: IEEE International Conference on Image Processing (ICIP), vol. 2, pp. 33–36. Barcelona, Spain (2003)
7. Brun, L., Kropatsch, W.: Contains and inside relationships within combinatorial pyramids. Pattern Recogni-

**Listing A.3** Adaptive access to topological links and embedding

```
MRHalfEdge* getOpposite(int level)
{
  if(level >= intro_level + opposite.size())
    return opposite.back();
  else
    return opposite[level - intro_level];
}

MRHalfEdge* getNext(int level)
{
  if(level >= intro_level + next.size())
    return next.back();
  else
    return next[level - intro_level];
}

MRHalfEdge* getEmbedding(int level)
{
  if(level >= intro_level + emb.size())
    return emb.back();
  else
    return emb[level - intro_level];
}
```

**Listing A.4** Algorithm for the traversal of the neighborhood of a vertex

```
MRHalfEdge * tmp = h;
do
{
  do something with
    tmp->getOpposite(l);

  tmp = tmp->getNext(l);
} while(tmp != h);
```

tion **39**(4), 515–526 (2006)
8. Catmull, E., Clark, J.: Recursively generated b-spline surfaces on arbitrary topological meshes. Computer-Aided Design **10**(6), 350–355 (1978)
9. Cazier, D., Dufourd, J.: A formal specification of geometric refinements. The Visual Computer **15**, 279–301 (1999)
10. DeFloriani, L., Kobbelt, L., Puppo, E.: A survey on data structures for level-of-detail models. Advances in Multiresolution for Geometric Modelling, Series in Mathematics and Visualization pp. 49–74 (2004)
11. Doo, D.: A subdivision algorithm for smoothing down irregularly shaped polygons. In: Proceedings on Interactive Techniques in Computer Aided Design, pp. 157–165. Bologna (1978)
12. Doo, D., Sabin, M.: Analysis of the behaviour of recursive division surfaces near extraordinary points. Computer-Aided Design **10**(6), 356–360 (1978)
13. Dufourd, J.F.: Algebras and formal specifications in geometric modeling. The Visual Computer **13**(3), 131–154 (1997)
14. Dyn, N., Levin, D., Gregory, J.A.: A butterfly subdivision scheme for surface interpolation with tension control. ACM Transactions on Graphics **9**(2), 160–169 (1990)
15. Edmonds, J.: A combinatorial representation for polyhedral surfaces. In: Notices of the American Mathematical Society, vol. 7 (1960)
16. Grasset-Simon, C., Damiand, G., Lienhardt, P.: nd generalized map pyramids: Definition, representations and basic operations. Pattern Recognition **39**(4), 527–538 (2006)
17. Herzen, B.V., Barr, A.: Accurate triangulations of deformed, intersecting surfaces. Computer Graphics **21**(4), 103–110 (1987)
18. Kobbelt, L.: $\sqrt{3}$ subdivision. In: Proceedings of SIGGRAPH'00, pp. 103–112 (2000)
19. Lee, A., Sweldens, W., Schröder, P., Cowsar, L., Dobkin, D.: Maps: Multiresolution adaptive parameterization of surfaces. In: Proceedings of SIGGRAPH'98, pp. 95–104 (1998)
20. Lee, M., Samet, H.: Navigating through triangle meshes implemented as linear quadtrees. ACM Transactions on Graphics **19**(2), 79–121 (2000)
21. Lienhardt, P.: Subdivision of n-dimensional spaces and n-dimensional generalized maps. In: 5th ACM Conference on Computational Geometry, pp. 228–236. Saarbrücken, Germany (1989)
22. Lienhardt, P.: Topological models for boundary representation: a comparison with n-dimensional generalized maps. Computer-Aided Design **23**(1), 59–82 (1991)
23. Loop, C.: Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah (1987)
24. Lounsberry, M., DeRose, T., Warren, J.: Multiresolution analysis for surfaces of arbitrary topological type. ACM Transactions on Graphics **16**(1), 34–73 (1997)
25. Pajarola, R.: Large scale terrain visualization using the restricted quadtree triangulation. In: Proceedings of the 9th IEEE Visualization '98, pp. 19–26. Research Triangle Park, USA (1998)
26. Pajarola, R., DeCoro, C.: Efficient implementation of real-time view-dependent multiresolution meshing. IEEE Transactions on Visualization and Computer Graphics **10**(3), 353–368 (2004)
27. Peters, J., Reif, U.: The simplest subdivision scheme for smoothing polyhedra. ACM Transactions on Graphics **16**(4), 420–431 (1997)
28. Peters, J., Shiue, L.J.: Combining 4- and 3-direction subdivision. ACM Transactions on Graphics **23**(4), 980–1003 (2004)
29. Schrack, G.: Finding neighbors of equal size in linear quadtrees and octrees in constant time. CVGIP: Image Understanding **55**(3), 221–230 (1992)
30. Shiue, L.J., Peters, J.: A mesh refinement library based on generic design. In: ACM-SE 43: Proceedings of the 43rd annual southeast regional conference, pp. 104–108. Kennesaw, Georgia, USA (2005)
31. Shiue, L.J., Peters, J.: A pattern-based data structure for manipulating meshes with regular regions. In: GI '05: Proceedings of the 2005 conference on Graphics interface, pp. 153–160. Victoria, British Columbia, Canada (2005)
32. Stam, J., Loop, C.: Quad/triangle subdivision. Computer Graphics Forum **22**(1), 79–85 (2003)
33. Taubin, G.: A signal processing approach to fair surface design. In: Proceedings of SIGGRAPH'95, pp. 351–358 (1995)
34. Taubin, G.: Detecting and reconstructing subdivision connectivity. The Visual Computer **18**(5-6), 357–367 (2002)
35. Vince, A.: Combinatorial maps. Journal of Combinatorial Theory pp. 1–21 (1983)
36. Weiler, K.: Edge-based data structures for modeling in curved-surface environments. IEEE Computer Graphics & Applications **5**(1), 21–40 (1985)
37. Zorin, D.: Modeling with multiresolution subdivision surfaces. Eurographics'05 Tutorial (2005)

38. Zorin, D., Schröder, P., DeRose, T., Kobbelt, L., Levin, A., Sweldens, W.: Subdivision for modeling and animation. SIGGRAPH'00 Course Notes (2000)
39. Zorin, D., Schröder, P., Sweldens, W.: Interpolating subdivision for meshes with arbitrary topology. In: Proceedings of SIGGRAPH'96, pp. 189–192 (1996)
40. Zorin, D., Schröder, P., Sweldens, W.: Interactive multiresolution mesh editing. In: Proceedings of SIGGRAPH'97, pp. 259–268 (1997)

**Pierre Kraemer** Pierre Kraemer is a PhD candidate in computer science at Strasbourg University, France and a member of LSIIT CNRS Lab since 2005. Before starting his PhD, he received his graduate Master degree in computer science from the Strasbourg University in 2005. His research interests include geometric modeling and geometry processing.

**David Cazier** David Cazier is an assistant professor of computer science at the Strasbourg University, France and a researcher at the LSIIT CNRS Lab since 1999. He received his PhD in computer science from the Strasbourg University in 1997. His research interests include formal specifications and computational geometry in geometric modeling.

**Dominique Bechmann** Dominique Bechmann is a professor of computer science at the Strasbourg University, France and a researcher at the LSIIT CNRS Lab since 1996. For 10 years now, she has also been head of the computer graphics research team in Strasbourg, France. From 1986 to 1989, she did her PhD at the IBM scientific center (Paris, France) on geometric modeling of anatomic organs. In 1990, she did a postdoc at the T.J. Watson research center (Yorktown Heigths, New York, USA) on deformation models. In 1991, she became an assistant professor and since then she is working on geometric modeling and virtual reality.