CPH: a compact representation for hierarchical meshes generated by primal refinement

Lionel Untereiner, Pierre Kraemer, David Cazier, Dominique Bechmann

Volume xx (200y), Number z, pp. 1–13

CPH: a compact representation for hierarchical meshes generated by primal refinement

L. Untereiner[2] and P. Kraemer[1] and D. Cazier[1] and D. Bechmann[1]

[1]Université de Strasbourg, ICube, CNRS, France
[2]Inria, Villers-lès-Nancy, F-54600, France

Abstract

We present CPH (Compact Primal Hierarchy): a compact representation of the hierarchical connectivity of surface and volume manifold meshes generated through primal subdivision refinements. CPH is consistently defined in several dimensions and supports multiple kinds of tessellations and refinements, whether regular or adaptive. The basic idea is to store only the finest mesh, encoded in a classical monoresolution structure that is enriched with a minimal set of labels. These labels allow traversal of any intermediate level of the mesh concurrently without having to extract it in an additional structure. Our structure allows attributes to be stored on the cells not only on the finest level, but also on any intermediate level. We study the trade-off between the memory cost of this compact representation and the time complexity of mesh traversals at any resolution level.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Object hierarchies—Boundary representations I.3.6 [Methodology and Techniques]: Graphics data structures and data types—

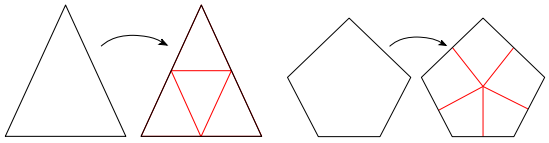
1. Introduction

In the past decade multiresolution representations have become very popular in the computer graphics community [DFS05]. In this context an object is represented as a hierarchy of nested meshes encoding different levels of detail. In the case of surface meshes, this representation gives access to multiresolution geometry processing tools like multiresolution edition, compression or signal processing. In the volumetric case, it is used for Free Form Deformation [CGC*02] or to speed-up mechanical simulation for the modelling of physical phenomena [BHU10].

Many of the algorithms used in such applications rely on an underlying topological structure that must be able to perform neighborhood queries on each level of the mesh hierarchy. Several kinds of tesselations (triangle, quad, tetrahedron, hexahedron, ...) must be supported by this structure as different algorithms may use different kinds of elements. Moreover, given the growing complexity of the objects handled by modern applications, the representation of the connectivity should be as compact as possible.

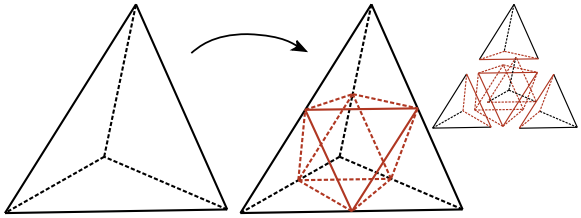
When the hierarchy is generated through a subdivision process, the design of the topological structure can benefit from the intrinsic regularity of the mesh. In this context, we propose CPH (*Compact Primal Hierarchy*), an innovative compact representation for mesh hierarchies generated by regular or adaptive primal refinements. It supports both surface and volume meshes and a wide variety of tessellations. The basic idea is to store only the finest version of the mesh in a standard monoresolution topological structure augmented with a minimal set of labels. These labels allow traversal of any of the coarser versions of the mesh without having to extract it into an additional data structure. Switching from one level to another, or even performing concurrent traversals at different levels of the mesh, is performed at no additional storage cost. Compared to a representation that would provide an explicit encoding of the connectivity at each intermediate level, the on-the-fly traversal of the different versions of the mesh is achieved at the expense of some time complexity overhead. We provide a precise analysis of both memory savings and time overhead in a theoretical and practical way.

The hierarchical meshes handled by the CPH representation result from the following process. Given a base mesh M^0 and a subdivision process S, the recursive process defined by $M^{i+1} = S \cdot M^i$ produces a hierarchy of *subdivision meshes*. For surfaces, primal refinements first insert a vertex into each edge and then split faces. All faces can be refined

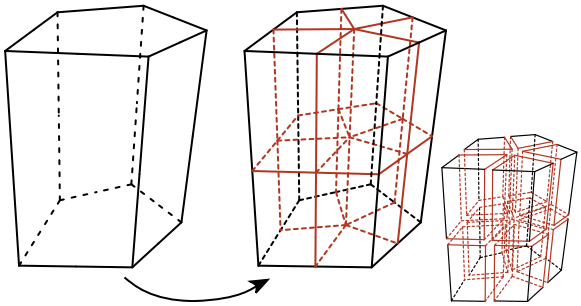


(a) 1-to-4 split of triangles (b) 1-to-n split of arbitrary polygons

Figure 1: 2-dimensional primal subdivision patterns.



(a) tetrahedral split



(b) 1-to-n split of arbitrary polyhedra.

Figure 2: 3-dimensional primal subdivision patterns.

using the 1-to-n split (figure 1b). Triangles can also be refined using the specific 1-to-4 split that preserves triangular faces (figure 1a). Both strategies are compatible within the same surface mesh. For volumes, primal refinements start by applying a primal surface refinement on each polyhedron and then split volumes. All volumes can be refined using the 1-to-n split (figure 2b). Tetrahedra can also be refined using the specific tetrahedron split (figure 2a).

The remainder of the paper is organized as follows. In section 2 we give an overview of existing structures for the representation of such hierarchies. Section 3 is dedicated to the presentation of the CPH representation. The management of attributes within the hierarchy is explained in section 4. Section 5 details the time complexity of the resolution level traversals both theoretically and in practical experiments. Finally, we propose in section 6 a practical compact encoding of the representation and compare its memory cost to that of existing structures.

2. Related work

A common approach for the representation of multiresolution meshes [DFM02, DFKP05] involves a coarse base mesh equipped with an additional hierarchical data structure.

A flexible strategy that has been defined for both cellular and simplicial complexes [DFPP97, DFPP99] consists in storing a set of local modifications in nodes whose dependencies are encoded in a directed acyclic graph (DAG). Access to a topological representation of a particular version of the mesh is achieved through the maintenance of an extracted mesh that is stored in an additional monoresolution structure and corresponds to an active front in the DAG. Each modification in the active front – like incrementing or decrementing the desired resolution level – gives rise to the application of the corresponding updates in the extracted mesh. Concurrent access to several versions or resolution levels leads to the maintenance of several fronts along with their associated extracted mesh.

Tree-based structures [Sam90, LS00] are more specialized structures adapted to a specific type of tesselation and refinement. The quadtree and octree data structures are naturally derived from the nested hierarchy of meshes generated by the application of a primal subdivision process on a surface or volume mesh. These structures allow traversal of the mesh that corresponds to a given resolution level without having to extract it into an additional mesh structure. However, not all the cells of the mesh (vertices, edges, faces, volumes) are explicitly represented, leading to difficulties with the management of attributes on these cells. As these structures are specialized for a given type of tesselation and refinement, they cannot be used to represent hierarchies in which several refinements are used, as in the quad/triangle subdivision [SL03].

[KCB09, UCB13] present another approach to handle multiresolution meshes based on a multiresolution extension of combinatorial maps. This representation stores the set of combinatorial maps that represent the mesh at each level of the hierarchy. Therefore it allows an efficient, direct and concurrent access to the topology of the mesh at any desired level. Its flexibility allows the support of a wide range of possible tessellations and refinements. However, as it explicitly stores the connectivity of the different resolution levels, its memory cost can be considered prohibitive in some application contexts.

Panozzo and Puppo [PP11] propose an adaptive subdivision surface scheme that computes the geometry of the vertices in an adaptively refined quad mesh consistently with the regular Catmull-Clark scheme. In order to be able to retrieve the stencils required for the computation of the geometry without maintaining a complex hierarchical structure, they propose a compact encoding of the mesh hierarchy in a standard monoresolution mesh data structure. Using a small amount of additional information, this structure allows traversal of any intermediate level without having to

extract it in an additional mesh data structure. However, as genericity is not their primary goal, this compact structure is restricted to quadrilateral surface meshes and does not support subdivision on triangular or volume meshes.

3. The CPH representation

Our objective is to design a structure that is as flexible as the representation proposed in [KCB09, UCB13] while being as compact as the one proposed in [PP11]. The CPH representation supports surface or volume meshes, any type of tessellation and any of the aforementioned subdivision schemes. Each level of the hierarchy is directly and concurrently available without having to extract it into an additional mesh data structure. A simple parameter determines the resolution level that is dynamically traversed by the proposed algorithms. Attributes are managed for any type of cell at any intermediate level.

For their flexibility and consistency in several dimensions, we decided to base the CPH representation on combinatorial maps, for which we give a short introduction.

3.1. Combinatorial maps

A mesh describes the cellular decomposition of a geometric object such as a curve, a surface or a volume. In the case of manifold object representation, many data structures have been proposed to encode the topological information of these meshes at different dimensions. For example, the half-edge [Wei85] data structure for surface meshes and the half-face data structure [LT97] for volume meshes are widely used.

These structures find their roots in the notion of *combinatorial maps* described in 1960 by Edmonds [Edm60]. It has been extended by Lienhardt to represent the subdivision of n-dimensional quasi-manifolds [Lie91]. An n-map M is a $n+1$-tuple $(D, \phi_1, \phi_2, \ldots, \phi_n)$ where D is a finite set of entities called darts and $\phi_1, \phi_2, \ldots, \phi_n$ are relations between these darts.

Darts linked with relation ϕ_1 form closed oriented cycles (top of figure 3). Relation ϕ_2 is a one-to-one relation between darts that links closed cycles pairwise through a common edge to form a surface mesh (middle of figure 3). Relation ϕ_3 is a one-to-one relation between darts that links surfaces pairwise through a common face to form a volume mesh (bottom of figure 3).

Unlike most data structures, cells of the subdivision (vertices, edges, faces, volumes, etc.) are not explicitly represented but rather defined as sets of darts. Figure 3 illustrates these different sets for 1, 2 and 3-dimensional maps. These sets of darts are formally defined as orbits on D. For example, in a 3-dimensional map, the face of d is the orbit $<\phi_1, \phi_3>(d)$ (in red in figure 3), i.e. the set of all darts that can be reached from d by applying ϕ_1 and ϕ_3 relations.

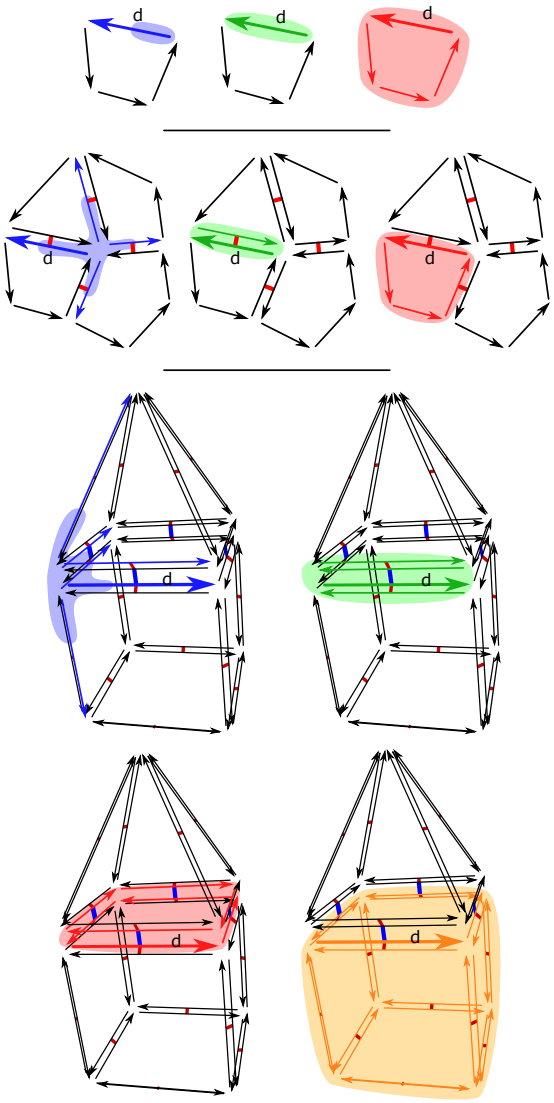


Figure 3: 1, 2 and 3-dimensional maps. Cells are represented by subsets of darts. In each case, the vertex of the dart d is highlighted in blue, its edge in green, its face in red and its volume in orange.

Each dart belongs exactly to one orbit of each dimension and thus represents simultaneously a vertex, an edge, a face and a volume of the mesh. Each cell is also equally represented by any of its darts. For example, the four green darts in the 3-dimensional map in figure 3 are all representatives of the same edge of the volume mesh. The actual cells can be explicited through an indexing process that relates them with their attributes. For each type of cell where attributes are desired, all the darts representing the same cell are associated with a common index. Clearly, the indexing of the cells of any dimension is completely optional. If the cells of one or

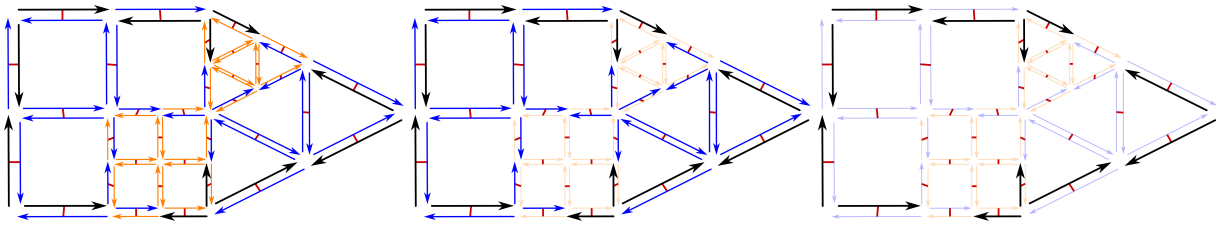


Figure 4: A 2-dimensional map has been adaptively subdivided twice. The insertion level of the darts is illustrated with colors: black, blue and yellow for darts inserted on levels 0, 1 and 2. The left figure illustrates the subset of darts D^2 defining the map of level 2. The subsets of darts D^1 and D^0 defining the maps of levels 1 and 0 are highlighted on the middle and right figures.

even all dimensions are not indexed, the cellular decomposition and its topology are still completely defined. Indeed, cell enumeration and neighborhood traversals are performed using exclusively the darts and their relations.

3.2. Objectives

The construction of the hierarchy starts with a coarse mesh represented by the combinatorial map $M^0 = (D^0, \phi_1^0, ..., \phi_n^0)$. At each subdivision step, the refinement operators add new darts to the map and modify their relations, thus defining the successive combinatorial maps $M^i = (D^i, \phi_1^i, ..., \phi_n^i)$.

The basic idea used in the CPH representation is to store only the finest combinatorial map $M^k = (D^k, \phi_1^k, ..., \phi_n^k)$. Given this fine combinatorial map, the challenge is to recover the intermediate versions M^i of this map using the least amount of additional information and without needing any additional data structure. For a given level i, we have to reconstruct two pieces of information: the set of darts D^i and the relations between these darts $\phi_1^i, ..., \phi_n^i$. With this information available, level i can be considered and directly traversed exactly like a classical combinatorial map.

3.3. Dart sets

The refinement process adds new darts at each subdivision step. At each level i, the map is then defined by D^i, the set of all darts inserted between levels 0 and i. To be able to reconstruct each D^i as a subset of D^k, darts are labeled with their insertion level. D^i is then defined by the set of all darts whose insertion level is inferior or equal to i. Figure 4 illustrates this principle with a small 2-dimensional example mesh.

The amount of information needed to encode this insertion level depends on the maximum number of desired resolution levels.

3.4. Dart relations

Now that we have defined D^i for each resolution level, the next step is to reconstruct the $\phi_1^i, ..., \phi_n^i$ relations on these darts.

3.4.1. 2-dimensional case

The first relation to reconstruct is ϕ_1^i that defines the oriented faces as cycles of darts within D^i. Figure 5 shows a detail extracted from figure 4. In this example, we try to traverse the map at level 0. The darts of D^0 are highlighted in bold black. Starting from dart `d1`, the next dart in this oriented face at level 0 is `d2`. The process that reconstructs the ϕ_1^0 relation has to jump over several darts to go from `d1` to `d2`.

The basic idea is to go through all vertices that were successively inserted in the original level 0 edge during the refinement process until finding a dart of the level being considered. For each intermediate vertex, some incident edges must be skipped to get back on the original edge and continue the path. The number of edges to skip depends on the refinement algorithm used in the incident face. In the face of `d1`, two edges were connected in each new vertex (triangular refinement). In the face of `d3`, a single edge was connected in each new vertex (quad refinement). In order for the algorithm to be generic and to know where to stop in the edge skipping process, the original edges are identified using labels.

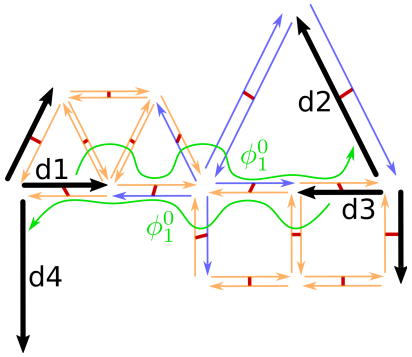


Figure 5: The ϕ_1^i relation is defined by a path in D^k between two darts of D^i that followed each other in a face of level i before it was subdivided. This path follows the original level i edge across the vertices that were inserted by the subdivision process.

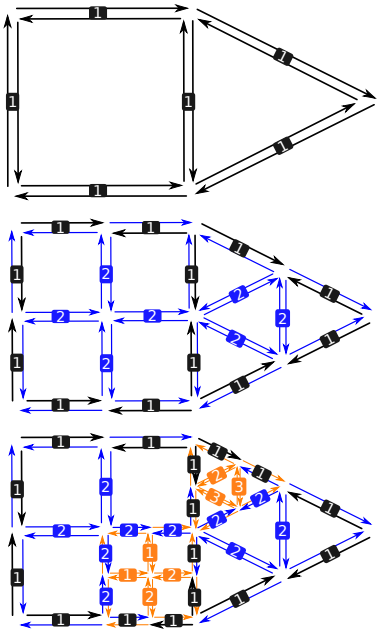


Figure 6: Edge labels are used as a guide to reconstruct ϕ_1^i relations. Only three different labels are needed to be able to distinguish an original edge from all edges connected to it during the refinement steps.

A first approach consists in assigning a unique edge label to each edge on the coarse mesh. At each refinement step, the label of a subdivided edge of level i is copied on the two resulting edges of level $i+1$. New labels are given to the new edges created by the subdivision process (shown in red in figure 1). The resulting labels can be used to identify the original edges of the mesh and thus successfully reconstruct the ϕ_1^i relations on each resolution level i. However, this approach has a strong limitation: the total number of uniquely identifiable edges is limited to the number of available labels. As our goal is to design a compact structure, the memory space used for these edge labels should be as small as possible.

In the ϕ_1^i reconstruction algorithm, the original edge only has to be distinguished from the ones that were connected in the intermediate vertices during the successive refinement steps. Thus, the label of each edge only needs to be different from the labels of the edges that are connected to it during the refinement. There is no need for a global label uniqueness. It follows that in the case of triangular 1-to-4 refinement, three different labels are sufficient. Indeed, each new edge connects two pre-existing edges from which it has to be distinguished. In the case of polygonal 1-to-n refinement, two different labels are sufficient as each new edge is connected to only one pre-existing edge. Figure 6 shows the edge labels used in the example mesh. At level 0, all labels are initialized to the same value. At each subdivision step, the label of a new edge is chosen as the smallest label different from those of the edges it connects.

In a map with k subdivision levels, the algorithm to compute $\phi_1^i(d)$ for a given dart d, with $i \in [0,k]$ is the following:

```
1:  function phi_1^i(d)
2:      finished ← false
3:      l ← edgeLabel(d)
4:      it ← d
5:      repeat
6:          it ← φ_1^k(it)
7:          if insertionLevel(it) ≤ i then
8:              ▷ DART OF D^i FOUND
9:              finished ← true
10:         else
11:             ▷ SKIP EDGES AROUND VERTEX
12:             while edgeLabel(it) ≠ l do
13:                 it ← φ_1^k(φ_2^k(it))
14:             end while
15:         end if
16:     until finished
17:     return it
18: end function
```

Let us point out that the only pieces of information used in this algorithm to reconstruct relation ϕ_1 at level i are the insertion level, the edge label and the relations between the darts on level k (lines 6 and 13).

The second relation to reconstruct is ϕ_2^i that links faces pairwise through a common edge. This relation can be defined from the ϕ_1^i relation. For a given dart d, $\phi_2(d)$ is the dart that represents the same edge as d but in the adjacent face. This dart is also part of the next vertex in the face of d. In the example of figure 7, $\phi_2^0(d1) = d3$. Going from $d1$ to $d3$ can be done by going first from $d1$ to $d2 = \phi_1^0(d1)$ which represents the next vertex in the face of $d1$ on level 0. Then, using relations on level k, from $d2$ to $d3 = \phi_2^k(\phi_{-1}^k(d2))$ which represents the next face around this vertex in counter-

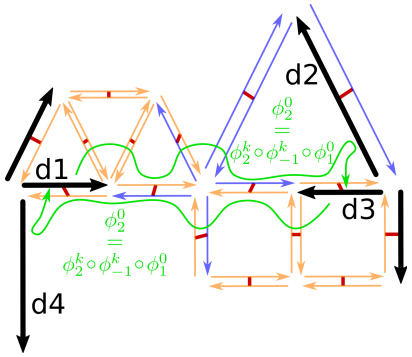


Figure 7: The ϕ_2^i relation is defined by the composition of the ϕ_1^i relation and a counter-clockwise turn around the reached vertex using level k relations.

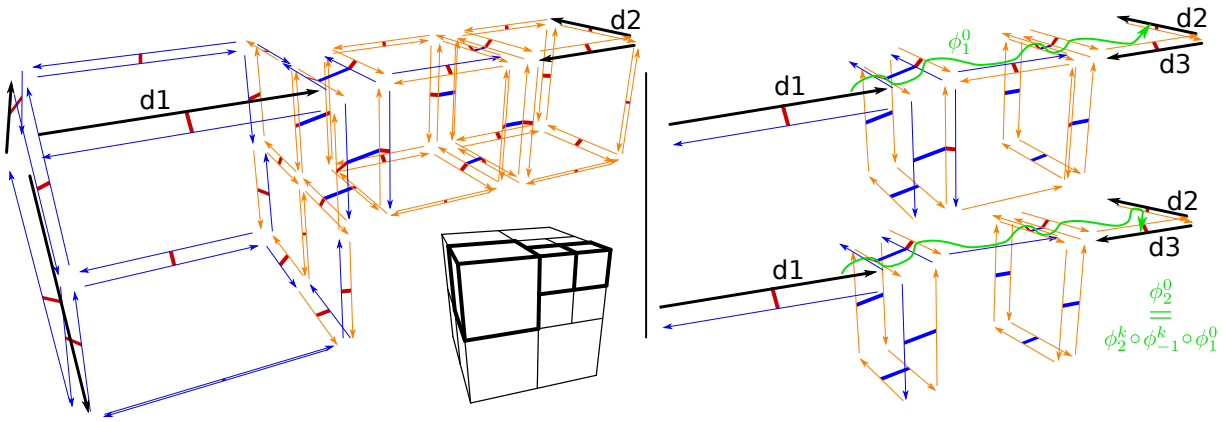


Figure 8: In a 3-dimensional map, the ϕ_1^i relation is also defined by a path in D^k between two darts of D^i that followed each other in a face of level i before its subdivision. The ϕ_2^i relation is defined using the ϕ_1^i relation and a counter-clockwise turn around the reached vertex.

clockwise order. The ϕ_{-1} relation used here is the inverse of the ϕ_1 relation, i.e. $\phi_{-1}(d)$ is the predecessor of d in the cycle formed by the ϕ_1 relation. For any level i and any dart $d \in D^i$, we have $\phi_2^i(d) = \phi_2^k(\phi_{-1}^k(\phi_1^i(d)))$.

With D^i, ϕ_1^i and ϕ_2^i, we have completely defined any intermediate resolution level as a map $M^i = (D^i, \phi_1^i, \phi_2^i)$. Thereby, just by choosing a value for $i \in [0,k]$, all the cells of the mesh of intermediate level i along with their neighborhood relations are available and can be traversed as in a classical combinatorial map.

3.4.2. 3-dimensional case

Similarly to the 2-dimensional case, the first relation to reconstruct is ϕ_1^i. Figure 8 shows a detail of an adaptively subdivided mesh. The original mesh (a cube) has been subdivided twice. The second step has been performed adaptively on only one volume of the level 1 mesh.

The basic idea is similar to the 2-dimensional case: go through all vertices that were successively inserted in the original edge during the refinement process until finding a dart of the level being considered. For each of these intermediate vertices, some incident edges must be skipped to get back on the original edge and continue the path. 3-dimensional refinement operations also add new faces. For each incident edge to skip, some incident faces must also be skipped to get back on the original face and continue the path. The number of edges and faces to skip depends on the refinement used in the volume. In order for the algorithm to be generic and to know where to stop in the edge and face skipping process, the original edges and faces are identified using labels. In the volume of `d1`, a single edge was connected to each new vertex (quad refinement) and a single face was connected to each new edge (hex refinement).

With the same compactness objective in mind, we extend the 2-dimensional optimal labeling to the 3-dimensional case. Primal volume refinements start by a surface refinement of each volume. Edge labels are thus treated in the exact same way as in the 2-dimensional case. In the case of polyhedral 1-to-n refinement, edges connected to the central vertex can take any label as they are not connected to any pre-existing edge.

The original faces must be distinguished from the ones that were connected to the intermediate edges during the successive refinements steps. Thus the label of each face must only be different from the labels of the faces that are connected to it during the refinement. It follows that in the case

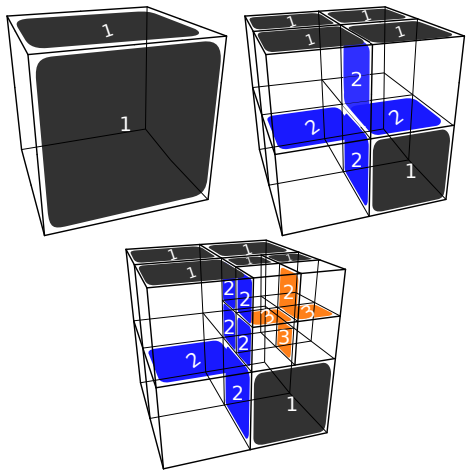


Figure 9: Some of the face labels are used as a guide to reconstruct ϕ_1^i relations. In the general case, only four different labels are needed to be able to distinguish an original face from all faces inserted in it during the refinement steps.

of tetrahedral refinement, four different face labels are sufficient. Indeed, each newly inserted face connects three existing faces from which it has to be distinguished. In the case of polyhedral 1-to-n refinement, three different labels are sufficient as each new face is connected to only two existing faces. At level 0, all labels are initialized to the same value. At each subdivision step, the label of a new face is chosen as the smallest label different from those of the faces it connects. Figure 9 shows some of the face labels used in the example mesh.

The 3-dimensional algorithm for $\phi_1^i(d)$, for a given dart d, is the following:

```
1:  function phi_1^i(d)
2:      finished ← false
3:      el ← edgeLabel(d)
4:      it ← d
5:      repeat
6:          it ← φ_1^k(it)
7:          if insertionLevel(it) ≤ i then
8:              ▷ DART OF D^i FOUND
9:              finished ← true
10:         else
11:             ▷ SKIP EDGES AROUND VERTEX
12:             while edgeLabel(it) ≠ el do
13:                 fl ← faceLabel(it)
14:                 it2 ← φ_2^k(it)
15:                 ▷ SKIP FACES AROUND EDGE
16:                 while faceLabel(it2) ≠ fl do
17:                     it2 ← φ_2^k(φ_3^k(it2))
18:                 end while
19:                 it ← φ_1^k(it2)
20:             end while
21:         end if
22:     until finished
23:     return it
24: end function
```

The ϕ_2^i definition given in the 2-dimensional case still stands in the 3-dimensional case. We have $\phi_2^i(d) = \phi_2^k(\phi_{-1}^k(\phi_1^i(d)))$. The ϕ_3^i definition is defined following the same approach, and we have $\phi_3^i(d) = \phi_3^k(\phi_{-1}^k(\phi_1^i(d)))$.

With D^i, ϕ_1^i, ϕ_2^i, ϕ_3^i, we have completely defined any intermediate resolution level as a map $M^i = (D^i, \phi_1^i, \phi_2^i, \phi_3^i)$. Again, just by choosing a value for $i \in [0,k]$, all the cells of the mesh of intermediate level i along with their neighborhood relations are available and can be traversed as in a classical combinatorial map.

4. Attribute management

As already mentioned in section 3.1, attributes can be associated to the cells of a combinatorial map through an indexing process. All the darts representing a same cell are associated with a common index that is used to retrieve the corresponding values in a set of attribute tables.

The CPH structure allows the association of attributes to each cell of the mesh at each intermediate level of the hierarchy. Due to the properties of primal refinements, vertex and other cell attributes are not managed in the same way.

4.1. Vertex attributes

Primal refinements leave vertices unchanged: no other cell is connected to an existing vertex. This means that the set of darts that defines a vertex never changes from its insertion level to the maximum level. Thus, a vertex is associated with a single index through all the levels of the hierarchy.

In order to associate different attributes with a vertex on each level where it is present in the mesh, the CPH structure can maintain an intermediate indirection table. For each vertex, this table contains a set of indices that point to the actual vertex attribute tables. The size of this set of indices is $(k-l)$ where k is the maximum level and l is the insertion level of the vertex. The attribute values of a vertex on level i are associated with the index stored in the $(i-l)$th element of this set. This mechanism is illustrated in figure 10.

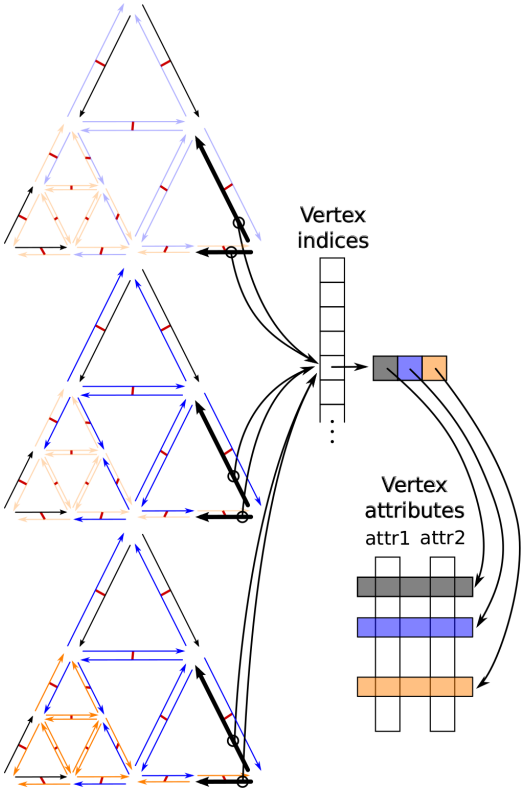


Figure 10: The darts of a vertex store only one index through all the levels of the hierarchy. This index refers to a cell of an indirection table that contains, for each level between the vertex insertion level and the maximum level, an index to the tables that contain actual vertex attribute values.

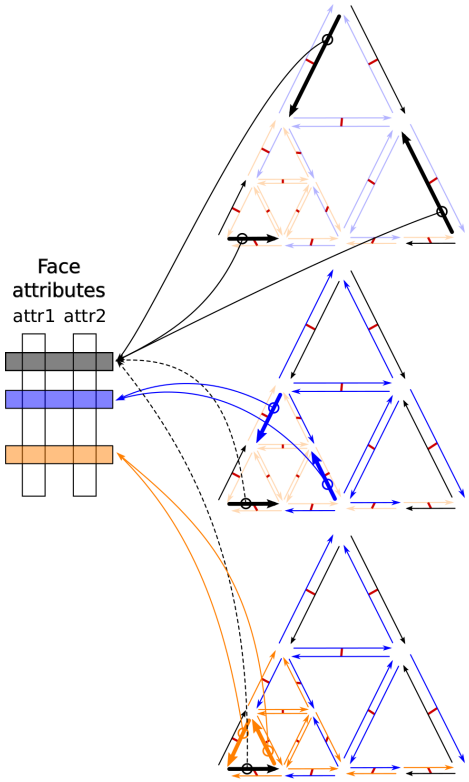


Figure 11: For higher dimensional cells, each dart keeps the index it has received on its insertion level through all levels of the hierarchy. Access to the attributes of a cell at a given level is then achieved by first retrieving the index associated to one of its most newly inserted darts. This figure illustrates face attribute indices for level 0 (black), level 1 (blue) and level 2 (orange). Plain arrows are the one that can be used to access faces attribute values.

For a given vertex on a given resolution level, access to its index and thus to its attribute values is done in constant time. However, compared to the cost of attribute management in a monoresolution structure, this feature induces a memory overhead, i.e. the indirection table. If an application only needs a single set of attribute values per vertex for every level where it is present in the mesh, the indirection table could be bypassed and its memory cost completely avoided.

4.2. Other cell attributes

Higher dimensional cells (edges, faces, volumes) are all split during a primal refinement step. The set of darts that define one of these cells is separated into several parts. Subdivided cells are composed of a mix of *old* and *new* darts. In this context, a mechanism that does not cost any additional memory can be proposed.

When a cell is refined, a new index is requested for each new cell. These indices are associated only with the *new* darts of these cells, leaving the indices associated with the *old* darts unchanged. Some new cells are composed only of *new* darts that are all associated with the new index (for example: interior triangle face and interior edges in 1-to-4 surface refinement, or interior faces and edges in 1-to-n volume refinement). The attribute values of a cell on level i are associated with the index stored on one of the *most newly* inserted darts of the cell. This mechanism is illustrated in figure 11.

Unlike vertex attribute management, this way of accessing cell attributes does not induce any memory overhead. However, as the access to the index requires finding one of the *most newly* inserted darts of the cell, the worst case access time is linear in the number of darts of the cell.

5. Reconstruction time analysis

When traversing an intermediate resolution level, most of the time is spent in the reconstruction of the ϕ_1^i relation. Indeed, the other ϕ_2^i and ϕ_3^i relations are defined based upon this one. We first provide a theoretical analysis of the reconstruction process of this relation. Then we show some benchmarks of our implementation of the CPH structure.

5.1. Theoretical analysis

Let us consider traversing the map at level $i, 0 \leq i \leq k$. The reconstruction process of the ϕ_1^i relation consists in following a path in M^k – the finest combinatorial map – between a dart $d \in D^i$ and $\phi_1^i(d) \in D^i$. The complexity of this process can be expressed by the length of this path. As presented in section 3.4, the idea used in the CPH representation is to follow the original edge and skip the vertices that were inserted in it on levels higher than i. The length of this path is thus directly related to the number of subdivision steps that has been applied to this edge in the higher levels.

In the case of a regular subdivision – which is the worst case – the edge of level i has been cut into 2^{k-i} edges in level k. This means that getting $\phi_1^0(d)$ – i.e. at resolution level 0 – costs 2^k more time in the CPH structure than in a data structure where this relation would be explicitly available. Besides, getting $\phi_1^k(d)$ – i.e. at resolution level k – does not cost anything more as these relations are the ones that are stored. At the same time, the number of cells of the mesh decreases along with the resolution level. Between two levels i and $i-1$, the cost of the reconstruction of ϕ_1^i is multiplied by 2, while the number of darts of the mesh is divided by 4. The cost of the reconstruction of the whole mesh is thus divided by 2 between levels i and $i-1$. Figure 12 illustrates the theoretical cost of the traversal of a mesh with the CPH structure and an explicit representation (i.e. where all relations are directly available on each level) as a function of the resolution level.

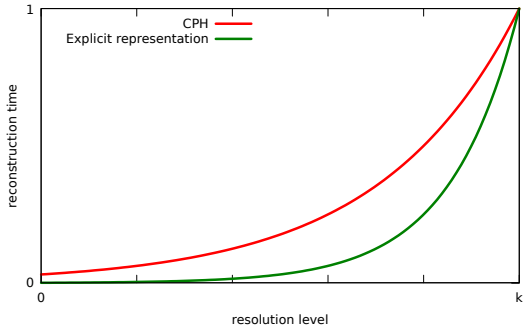


Figure 12: Comparison of the theoretical costs of the traversal of a regularly subdivided mesh with a CPH structure and with an explicit representation, as a function of the resolution level.

Many geometry processing or computation algorithms built over a hierarchical mesh perform successive traversals of all resolution levels. Thus, we further extend our analysis to this setting.

In the case of regular triangular refinement, the number of darts on each resolution level i is $d_0 \cdot 4^i$, with d_0 the number of darts on the coarse level 0 mesh. The cost of traversing all ϕ_1^i relations is thus $d_0 \cdot 4^i \cdot 2^{k-i}$. Considering the successive traversals of all resolution levels, the total cost is:

$$CPH = \sum_{i=0}^{k} d_0 \cdot 4^i \cdot 2^{k-i} = d_0 \cdot \sum_{i=0}^{k} 2^{k+i} = d_0 \cdot 2^k \cdot \sum_{i=0}^{k} 2^i$$

By identifying the above sum with $\sum_{i=0}^{n} x^i = \frac{x^{n+1}-1}{x-1}$ it follows:

$$CPH = d_0 \cdot 2^k \cdot 2^{k+1} = d_0 \cdot 2^{2k+1}$$

In an explicit representation, as ϕ_1^i relations are traversed in constant time, the cost of the same traversal is the following:

$$EXPL = \sum_{i=0}^{k} d_0 \cdot 4^i = d_0 \cdot \sum_{i=0}^{k} 4^i$$

Using the same identification, we obtain:

$$EXPL = d_0 \cdot \frac{4^{k+1}}{3} = d_0 \cdot \frac{2^{2k+2}}{3}$$

The ratio between the cost of CPH and an explicit representation is thus:

$$\frac{CPH}{EXPL} = \frac{d_0 \cdot 2^{2k+1}}{d_0 \cdot \frac{2^{2k+2}}{3}} = \frac{3}{2}$$

In the case of regular hexahedral mesh refinement, the number of darts on each resolution level i is $d_0 \cdot 8^i$, with d_0 the number of darts on the coarse level 0 mesh. As above, the cost of traversing all ϕ_1^i relations is thus $d_0 \cdot 8^i \cdot 2^{k-i}$. Considering the successive traversals of all resolution levels, the total cost is:

$$\begin{aligned} CPH &= \sum_{i=0}^{k} d_0 \cdot 8^i \cdot 2^{k-i} \\ &= d_0 \cdot \sum_{i=0}^{k} 2^{k+2i} \\ &= d_0 \cdot 2^k \cdot \sum_{i=0}^{k} 4^i \\ &= d_0 \cdot 2^k \cdot \frac{4^{k+1}}{3} \\ &= d_0 \cdot \frac{1}{3} \cdot 2^{3k+2} \end{aligned}$$

In an explicit representation, the cost of the same traversal is the following:

$$EXPL = d_0 \cdot \sum_{i=0}^{k} 8^i = d_0 \cdot \frac{8^{k+1}}{7} = d_0 \cdot \frac{1}{7} \cdot 2^{3k+3}$$

The ratio between the cost of CPH and an explicit representation is thus:

$$\frac{CPH}{EXPL} = \frac{d_0 \cdot \frac{1}{3} \cdot 2^{3k+2}}{d_0 \cdot \frac{1}{7} \cdot 2^{3k+3}} = \frac{7}{6}$$

An important property of the CPH representation is that these ratios are *constant* in both cases and depend neither on the complexity of the initial coarse mesh nor on the number of resolution levels in the hierarchy.

In the case of an adaptive subdivision, the number of subdivision steps applied to an edge between its insertion level and the maximum level varies throughout the mesh. In any case, the total cost of the traversal of an adaptively subdivided mesh will always be between that of an explicit representation and that of a CPH representation with a regularly subdivided mesh.

5.2. Practical experiments

We carried out some practical evaluations of our implementation of the CPH representation. We measured execution times with and without the numerical computation in order to highlight the specific cost of the mesh traversal. The first application shown in figure 13 uses a multigrid method to compute a harmonic function that satisfies given boundary conditions over a triangular mesh using a discrete Laplace-Beltrami operator. The second application is a multigrid electro-thermal simulation that is run over hexahedral meshes (see figure 14). In both cases, the algorithms go through all the resolution levels successively and perform computations that exploit local connectivity information on each intermediate level.

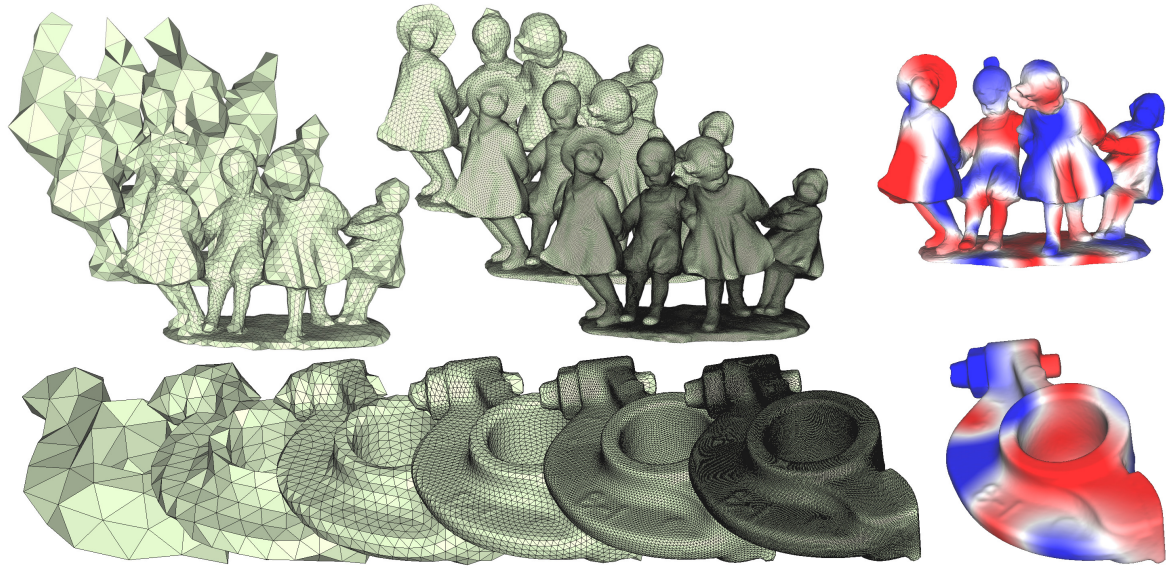


Figure 13: Practical experiments using the CPH model on surfaces: harmonic function computation on triangular meshes.

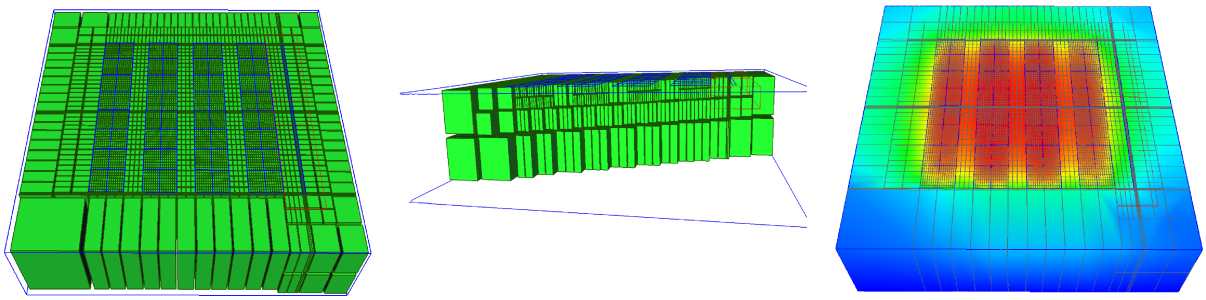


Figure 14: Practical experiments using the CPH model on volume meshes: electro-thermal simulation on an adaptively subdivided hexahedral mesh.

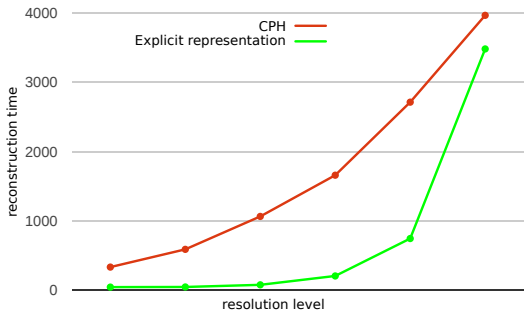


Figure 15: Practical comparison of the cost of the traversal of a regularly subdivided mesh with a CPH structure and an explicit representation. Our implementation reproduces what is expected from the theoretical study.

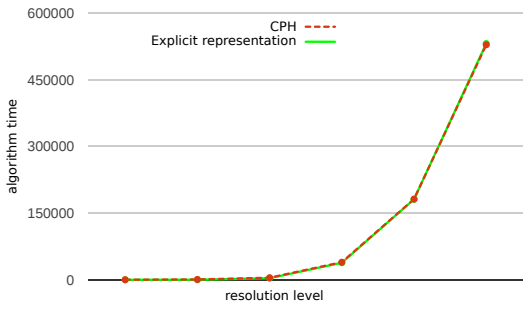


Figure 16: Practical comparison of the cost of a multigrid computation on a regularly subdivided mesh with a CPH structure and an explicit representation. The overhead due to the on-the-fly reconstruction of the intermediate levels in the CPH structure is here negligible.

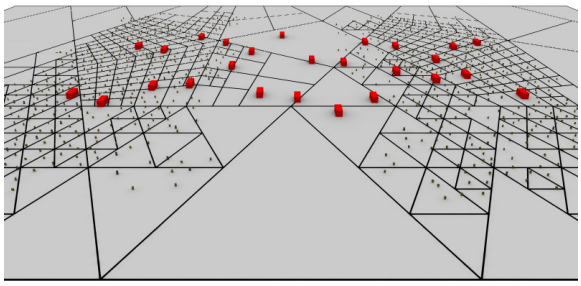


Figure 17: A crowd simulation framework that uses the CPH structure for the representation of the environment. Blacks dots and red boxes represent the agents populating the simulation. The underlying environment is used as an accelerating structure for proximity queries and is dynamically subdivided based on the local density of the crowd.

Figure 15 shows the performances of the mesh traversal algorithms (i.e. without any additional process). It can be observed that these practical timings reproduce the theoretical results obtained in the previous section (figure 12). Figure 16 shows the cost of the entire process for each level. As the cost of the numerical computation is independent of the underlying mesh representation and is more important than that of the mesh traversal, the overhead due to the dynamic reconstruction of the intermediate levels in the CPH structure is here negligible.

The CPH structure is particularly well adapted to applications that need a hierarchical structure where the main level of interest is the finest level of the mesh. This is the case of the crowd simulation framework presented in [JKC12] that uses the CPH structure to represent the environment. The mesh is adaptively refined and dynamically updated at each time step based on the local crowd density. The finest level is used as an accelerating structure for proximity queries, which are performed at each time step. Coarser levels are used for path planning, which is only performed when needed. Figure 17 shows an image extracted from such a simulation.

6. Memory cost

The goal of the CPH representation is to avoid the cost of hierarchical information and thus to be as compact as possible compared to a more explicit formulation. As presented in section 3, defining and traversing the intermediate levels of the mesh requires storing the insertion level of a dart and labels for edges and faces (only in dimension 3).

In dimension 2, triangular 1-to-4 refinement needs 3 different edge labels and polygonal 1-to-n refinement needs only 2. This means that in the general case 2 bits are needed to store edge labels. If only 1-to-n refinement is used, a single bit can be used to store edge labels.

In dimension 3, tetrahedral refinement needs 3 different edge labels and 4 different face labels. The polyhedral 1-to-n refinement needs 2 different edge labels and 3 different face labels. In the first case, 2 bits are needed for edge labels and 2 bits for face labels. In the second case, 1 bit is needed for edge labels and 2 for face labels.

Table 1 summarizes this information and shows how many bits are left to encode the darts' insertion level if one chose to store all the additional information (i.e. labels and insertion level) in only 1 byte per dart. It is to be noted that in the worst case (tetrahedral refinement on volume meshes), 4 bits are left to encode the insertion level, which still allows encoding of 16 resolution levels.

Dimension	Dart information	Refinement	
		Quad	Tri or Tri/Quad
2-maps	Edge label	1 bit	2 bits
	Insertion level	7 bits	6 bits
3-maps	Edge label	1 bit	2 bits
	Face label	2 bits	2 bits
	Insertion level	5 bits	4 bits

Table 1: Summary of the memory cost of the additional information maintained by the CPH structure for 2-maps and 3-maps on a 1 byte per dart basis.

In order to compare the memory cost of the CPH structure, a hierarchical data structure that has a similar representation domain is needed. Like the CPH representation, it should be able to encode equivalent topological information and to manage attributes for any type of cell at any intermediate level of the hierarchy. We decided to compare the CPH structure to multiresolution maps [KCB09,UCB13], an explicit hierarchical mesh structure for surface (multiresolution 2-maps) and volume (multiresolution 3-maps) representation.

First, we compare the memory cost in the case of a regular triangle mesh refinement. Let d_0 be the number of darts at level 0 and k the number of resolution levels. A multiresolution 2-map stores a 2-map for each level of the hierarchy. The total number of darts is expressed by the following sum: $\sum_{i=0}^{k} d_0 \cdot 4^i$. This sum can be identified with $\sum_{i=0}^{n} x^i = \frac{x^{n+1}-1}{x-1}$ and simplifies to: $\frac{4}{3} \cdot d_0 \cdot 4^k$. Each dart stores two 4-byte indices (ϕ_1 and ϕ_2) plus one additional byte for the insertion level. The memory cost is: $EXPL = 12 \cdot d_0 \cdot 4^k$ bytes.

The CPH structure encodes an equivalent mesh in a monoresolution 2-map composed of $d_0 \cdot 4^k$ darts. Each dart stores two 4 byte indices (ϕ_1 and ϕ_2) plus one additional byte for labels and insertion level encoding (as presented in table 1). The memory cost is: $IMPL = 9 \cdot d_0 \cdot 4^k$ bytes. The ratio between the two data structures is therefore:

$$\frac{EXPL}{IMPL} = \frac{4}{3} = 1.33$$

In other words a multiresolution 2-map needs 33% more memory space than the CPH structure.

Next, we study the memory cost in the 3-dimensional case of regular hexahedral mesh refinement. Let d_0 be the number of darts at level 0 and k the number of resolution levels. A multiresolution 3-map stores a 3-map for each resolution level. The total number of darts is therefore expressed by the following sum: $\sum_{i=0}^{k} d_0 \cdot 8^i$. Using the same identification as above, this sum simplifies to: $\frac{8}{7} \cdot d_0 \cdot 8^k$. Each dart stores three 4-byte indices (ϕ_1, ϕ_2 and ϕ_3) plus one additional byte for the insertion level. The memory cost is: $EXPL = \frac{104}{7} \cdot d_0 \cdot 8^k$ bytes.

The CPH structure encodes an equivalent mesh in a monoresolution 3-map composed of $d_0 \cdot 8^k$ darts. Each dart stores three 4-byte indices (ϕ_1, ϕ_2 and ϕ_3) plus one additional byte for labels and insertion level encoding. The memory cost is: $IMPL = 13 \cdot d_0 \cdot 8^k$ bytes. The ratio between the two data structures is therefore:

$$\frac{EXPL}{IMPL} = \frac{104}{91} = 1.14$$

In other words, a multiresolution 3-map needs 14% more memory space than the CPH structure.

More memory space can be saved in the particular case of a regular subdivision of meshes composed of only one tessellation type (e.g. triangles, quadrangles, tetrahedrons or hexahedrons). In this case, darts may be allocated by blocks corresponding to the fixed combinatorics of the cells. For example, in the case of triangle meshes, darts can be allocated by groups of 3 and the ϕ_1 permutation can be reconstructed with simple arithmetic operations on dart indices.

7. Conclusion and future work

We have presented the CPH structure for the compact representation of mesh hierarchies generated by regular or adaptive primal refinement. As it is defined upon the combinatorial maps model, the CPH representation is consistently defined in several dimensions and supports multiple kinds of tessellation and refinement. The finest mesh is the only one to be stored and each intermediate level of the hierarchy is traversed on-the-fly thanks to a small number of labels that can be stored with only one additional byte per dart. A mechanism that allows storage of cell attributes on each level of the hierarchy is also described. The memory cost of CPH is lower than data structures that explicitly store the different levels such as multiresolution combinatorial maps. Also, as there is no need to extract an intermediate mesh in an additional structure in order to traverse it, the memory cost does not grow with the number of concurrently accessed levels. Finally, the time complexity overhead depends on neither the size of the base mesh nor the number of resolution levels.

Future work includes the management of a richer set of refinement operations such as dual refinement schemes. The application of our method to the context of progressive meshes is also possible and could be investigated. Starting from a given fine mesh, each edge collapse could be encoded by labelling the darts of the pair of deleted triangles. A similar algorithm to reconstruct relations between darts may then allow one to traverse an intermediate mesh by jumping over the deactivated cells. This idea needs further work that is beyond the scope of this article.

References

[BHU10] BURKHART D., HAMANN B., UMLAUF G.: Isogeometric finite element analysis based on catmull-clark subdivision solids. *Computer Graphics Forum 29*, 5 (2010), 1575–1584. 1

[CGC*02] CAPELL S., GREEN S., CURLESS B., DUCHAMP T., POPOVIĆ Z.: A multiresolution framework for dynamic deformations. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation* (New York, NY, USA, 2002), SCA '02, ACM, pp. 41–47. 1

[DFKP05] DE FLORIANI L., KOBBELT L., PUPPO E.: A survey on data structures for level-of-detail models. In *Advances in Multiresolution for Geometric Modelling*, Dodgson N. A., Floater M. S., Sabin M. A., (Eds.), Mathematics and Visualization. Springer Berlin Heidelberg, 2005, pp. 49–74. 2

[DFM02] DE FLORIANI L., MAGILLO P.: Multi-resolution mesh representation: Models and data structures. In *Tutorials on Multiresolution in Geometric Modelling*, Floater M. S., Iske A., Quak E., (Eds.), Mathematics and Visualization. Springer Berlin Heidelberg, 2002, pp. 363–417. 2

[DFPP97] DE FLORIANI L., PUPPO E., PAOLA M.: A formal approach to multiresolution hypersurface modeling. In *Geometric Modeling: Theory and Practice*, R. Klein R., Straßer W., Rau R., (Eds.), Focus on Computer Graphics. Springer Berlin Heidelberg, 1997, pp. 302–323. 2

[DFPP99] DE FLORIANI L., PAOLA M., PUPPO E.: Multiresolution representation of shapes based on cell complexes. In *Proceedings of the 8th conference on Discrete Geometry for Computer Imagery* (1999), DGCI '99, Springer Berlin Heidelberg, pp. 3–18. 2

[DFS05] DODGSON N. A., FLOATER M. S., SABIN M. A.: *Advances in multiresolution for geometric modelling*. Mathematics and visualization. Springer-Verlag Berlin Heidelberg, 2005. 1

[Edm60] EDMONDS J. R.: A combinatorial representation for polyhedral surfaces. In *Notices of the American Mathematical Society* (1960), vol. 7. 3

[JKC12] JUND T., KRAEMER P., CAZIER D.: A unified structure for crowd simulation. *Comput. Animat. Virtual Worlds 23*, 3-4 (May 2012), 311–320. 11

[KCB09] KRAEMER P., CAZIER D., BECHMANN D.: Extension of half-edges for the representation of multiresolution subdivision surfaces. *The Visual Computer 25*, 2 (2009), 149–163. 2, 3, 11

[Lie91] LIENHARDT P.: Topological models for boundary representation: a comparison with n-dimensional generalized maps. *Computer-Aided Design 23*, 1 (1991), 59–82. 3

[LS00] LEE M., SAMET H.: Navigating through triangle meshes implemented as linear quadtrees. *ACM Trans. Graph. 19*, 2 (Apr. 2000), 79–121. 2

[LT97] LOPES H., TAVARES G.: Structural operators for modeling 3-manifolds. In *Proceedings of the fourth ACM symposium*

on Solid modeling and applications (New York, NY, USA, 1997), SMA '97, ACM, pp. 10–18. 3

[PP11] PANOZZO D., PUPPO E.: Implicit hierarchical quad-dominant meshes. *Computer Graphics Forum 30*, 6 (2011), 1617–1629. 2, 3

[Sam90] SAMET H.: *The Design and Analysis of Spatial Data Structures*, vol. 50255 of *Addison-Wesley Series in Computer Science*. Addison-Wesley Publishing Company, 1990. 2

[SL03] STAM J., LOOP C.: Quad/triangle subdivision. *Computer Graphics Forum 22*, 1 (2003), 79–85. 2

[UCB13] UNTEREINER L., CAZIER D., BECHMANN D.: *n*-dimensional multiresolution representation of subdivision meshes with arbitrary topology. *Graphical Models 75*, 5 (2013), 231–246. 2, 3, 11

[Wei85] WEILER K.: Edge-based data structures for solid modeling in curved-surface environments. *Computer Graphics and Applications, IEEE 5*, 1 (1985), 21–40. 3