Journées du Groupe de Travail en Modélisation Géométrique 2018, Aix-en-Provence

Rasterized 3D Combinatorial Maps

Aldo Gonzalez-Lorenzo [1], Guillaume Damiand [1], Florent Dupont [1], and Jarek Rossignac [2]

[1]Univ Lyon, CNRS, LIRIS UMR5205, F-69622, France
[2]School of Interative Computing, Georgia Institute of Technology, Atlanta, USA

Résumé

Les cartes combinatoires sont une structure de données pour représenter la topologie d'objets subdivisés en cellules. Elles permettent de décrire et manipuler des objets sans aucune contrainte sur le type de cellules, mais elles sont coûteuses en mémoire. Le rasterized planar face complex *(rPFC) est une structure de données compacte récemment introduite qui permet d'encoder une carte combinatoire 2D dans une image tout en conservant sa topologie.*

Dans cet article nous explorons comment étendre cette structure de données en trois dimensions. Nous redéfinissons le rPFC sous un nouveau formalisme qui permet cette généralisation et nous discutons plusieurs implémentations fournissant différents compromis temps/espace mémoire.

Mots-clés : Volumic mesh, Combinatorial maps, Compact representation, Topology preserving rasterization

1. Introduction

While there exist compact and efficient data structures for surface, tetrahedral and hexahedral meshes [Wei85, Män88, dB00, GS85, Wei88, Ros01, DL87, Bri89, DRE*15], representing a volumetric object subdivided in volumes with arbitrary topology remains a challenging task.

The type of objects we wish to treat is a 3D oriented quasi-manifold, that is a subdivision of the 3D space in volumes (or polyhedra) that are glued by their faces. An explicit way of representing such object would be to index the vertices (pointing to their coordinates), the facets (or polygons), which are ordered sequences of vertices, and the volumes, which are (unordered) sequences of facets. The incidence relation between the cells is implicitly described in this structure, but its computation has linear complexity in the number of elements in the object, so we should explicitly represent at least some parts of it. If the number of faces and cofaces is not fixed, the overhead in space memory for this explicit representation can become enormous.

A different approach for representing a 3D quasi-manifold is the *3D combinatorial map* [DL14]. Instead of distinguishing between the cells and their incidence relations, both concepts are merged using *darts* and three operators β_i ($i \in [1,3]$) between darts. Though this approach may seem intricate at first, it actually reduces the representation of the cells and their incidence relation by encoding them as three arrays of integers (plus sometimes one more array for mapping each dart to its vertex coordinates). The weakness of this approach is the large number of darts compared to the number of cells and its resulting high memory cost, which prevents us from processing huge models.

The motivation for this work is to deal with large and topologically complex 3D models. In order to conceive a compact representation, we encode a 3D combinatorial map as a 3D picture (a grid of voxels), where each voxel contains a piece of information that allows the traversal of its darts. This was achieved for the two-dimensional case in a recent work by Damiand and Rossignac [DR17] with the so-called *rasterized Planar Face Complex* (rPFC). This data structure encodes a 2D combinatorial map (or equivalently, a half-edge data structure) in an image, where each pixel contains a word using 6 different symbols that encodes the intersections of the 2-map with the boundary of the pixel.

Intuitively, if we intersect the neighborhood of a vertex in a planar graph with a circumference (a square, in our case), each edge becomes a vertex and each face becomes an edge. We can retrieve the original topology of the neighborhood from this information, and the position of the vertex is bounded by the area of the circumference. The rPFC is based on this idea, but it also supports several vertices in the same pixel (as long as they are not connected by an edge) and edges that do not have an endpoint inside the pixel.

This paper presents our current work on extending the rPFC to three dimensions, which we call *rasterized 3D combinatorial map*, or 3-rmap for short. The main idea can be seen as follows: we intersect the neighborhood of each vertex with the boundary of a voxel, which is a 2-manifold and can be encoded with a rPFC. We show that we can achieve this by generalizing the concepts already present in the rPFC except for two points: (1) there is one extra condition on the intersection between the 3D combinatorial map and the voxel grid; and (2) there are two new symbols in the words. Moreover, these conditions are hard to satisfy in practice (for the 3D models that we have tested), so it is necessary to consider a hierarchical voxel grid instead of a regular grid.

The paper is organized as follows. In Section 2, we review the definition of the 3D combinatorial map and the rasterized planar face complex. In Section 3, we introduce the rasterized 3D combinatorial map. In Section 4, we discuss four implementations with different trade-offs between memory space and execution time and illustrate them with two examples. Section 5 concludes.

2. Preliminaries

2.1. 3D combinatorial map

A three-dimensional combinatorial map (or 3-map, for short) is a tuple $(D, \beta_1, \beta_2, \beta_3)$ such that (1) D is a finite set of darts; (2) β_1 is a permutation on D; (3) β_2 and β_3 are involutions on D; and (4) $\beta_1 \circ \beta_3$ is an involution. It represents a subdivision of a closed orientable 3-dimensional space. We usually denote $\beta_0 := (\beta_1)^{-1}$.

Each dart $d \in D$ corresponds to a sequence of incident cells (v, e, f, c), where v is a vertex (or 0-cell), e is an edge (1-cell), f is a face (2-cell) and c is a volume (3-cell). There are much more darts than cells in a 3-map, but this representation encodes the boundary and coboundary relation between the cells and can be used for cells of arbitrary topology (number of faces and cofaces).

A dart can be represented as an arrow over its edge, with the source being its vertex. The face of the dart is unequivocally determined by the orientation of the faces of the volume. Figure 1(left) illustrates a tetrahedron and 6 of its 12 darts.

The operator β_1 maps a dart to the next dart in the same face and volume, following its orientation. Hence, it only changes its vertex and its edge. The operator β_2 switches the face to the other face of the same volume that is incident to the edge. It also switches the vertex to respect the orientation of the face. Finally, β_3 switches the volume incident to the face and the vertex. Thus, each β_i only affects the vertex and the i-cell of the dart. The three operators are illustrated in Figure 1(right).

A 3-map is a topological description, without any information about the embedding of the cells. In this paper, we assume that the vertices are assigned coordinates in $\mathbb{R}^3$ and the cells are linear interpolations of its vertices. We assume that the cells only intersect themselves along lower dimension cells.

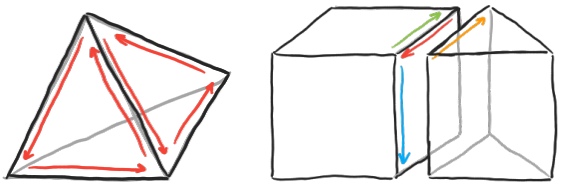

Figure 1: *Left: a tetrahedron (black) and six of its darts (red). Each dart has been moved inside its face for better visibility. Right: a 3-map with two volumes (separated to better see the different darts). The darts represented are d (red), $\beta_1(d)$ (blue), $\beta_2(d)$ (green) and $\beta_3(d)$ (orange).*

2.2. Rasterized planar face complex

The *rasterized Planar Face Complex* (rPFC) is a compact representation for 2D combinatorial maps (or equivalently, half-edge data structure or doubly connected edge list) introduced by Damiand and Rossignac in [DR17]. The main idea is to intersect the 2-map with a grid of pixels and derive a sequence of symbols for each pixel that allows the reconstruction of its topology, while the location of the vertices is bounded by the size of the pixels.

Given a 2-map, we consider a regular grid of pixels. We assume that (1) there is no intersection between the edges (resp., vertices) of the 2-map and the vertices (resp., edges) of the pixels and (2) no edge of the 2-map is contained in a single pixel.

Given a pixel $P = (i, j)$, we denote each intersection of an edge of the 2-map with any of its four faces a *crossing*. Each crossing corresponds to an edge having a vertex inside the pixel (type 0) or not (type 1). In the first case, there may be other crossings related to the same vertex inside the pixel. In the second case, there is exactly one other crossing for the same edge. Note that there is a bijection between the darts of the 2-map and the crossings of type 0 in the rPFC.

We sort the crossings following the clock-wise orientation of the boundary of the pixel starting from the bottom-left vertex of the pixel and we assign a symbol to each crossing c in the following way:

- Let c be of type 0. If it is the first crossing related to the vertex, its symbol is '`(`'. If it is the last, it is '`)`'. Otherwise, its is '`Y`'. If it is the only edge incident to that vertex, its symbol is '`=`'.
- Let c be of type 1. Its symbol is '`[`' if it is the first crossings, or '`]`' if it is the second one.

Hence, we define the *pixel-word* associated to the pixel P as

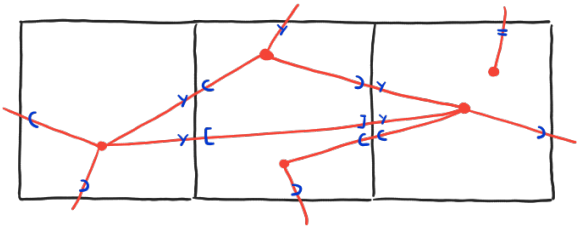

Figure 2: *Three consecutive pixels with words "(++YY+)+", "[(+Y+)Y](+)+" and "(YY+=+)++".*

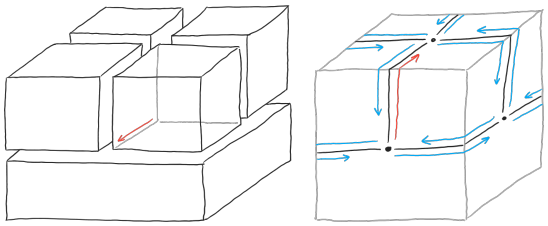

Figure 3: *A subdivided object (left) and the 2-map representation of its intersection with a voxel (right). The darts corresponding to the second crossing of the word of the front pixel are shown in red.*

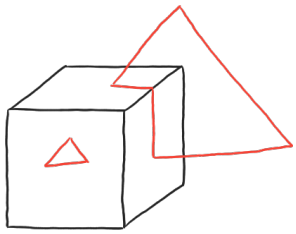

Figure 4: *A triangle having a non-connected intersection with the boundary of a voxel. If such a configuration exists between a 3-map and a voxel grid, then condition **C3** is not satisfied and they are not in general position.*

the concatenation of the symbols of its crossings, with four additional symbols '+' to separate each edge of the pixel. Note that, since the underlying graph is planar, the pixel-word is well parenthesized. We can identify a crossing in a pixel by its pixel P and its position i in its pixel-word. To illustrate this, Figure 2 depicts three pixels with their respective pixel-words. The crossings encoding the edges of the triangular face in the middle are (1,1;4), (1,2;6) and (1,3;2).

There are three basic operations defined on the rPFC. Let (P,i) be a crossing. $next(P,i)$ returns the previous crossing related to the same vertex. Similarly, $previous(P,i)$ returns the next crossing related to the same vertex. Finally, $convert(P,i)$ returns the same crossing in its adjacent pixel. By combining these three operations, we can obtain the operators β_1 and β_2 of the 2-map.

Note that we can use a single closing symbol instead of two, and then figure out the type of the crossing by examining the pixel-word. In all, the rPFC consists of a matrix of (usually short) words using only 6 symbols, which can be encoded using an entropy coding (such as a Huffman or an arithmetic coding) for compactness.

We refer the reader to the original work [DR17] for further details, notably on the algorithms for the functions $next$, $previous$ and $convert$.

3. Definitions for the Rasterized 3D Combinatorial Map

Consider the subdivided object in Figure 3(left). By intersecting it with the boundary of a voxel placed at its center, we obtain a planar graph embedded in the boundary of the voxel, depicted in Figure 3(right). It has 5 vertices, 8 edges and 5 faces. Note that each of these i-cells corresponds to an $(i+1)$-cell of the subdivided object. This graph can be represented as a 2D combinatorial map and even as a rPFC. For instance, the pixel-word of the front face is "(+Y+)++". The crossing in the top edge corresponds to the dart $d=(v,e,f)$, where e is the edge making the crossing and v is the vertex contained in the pixel. Its face f can only be the face on the right side of e. Moreover, the same crossing (in this voxel) corresponds to a dart in the subdivided object. The i-cell ($i \geq 1$) of the dart is the cell identified with the $(i-1)$-cell of the dart d in the 2-map. Its vertex is the vertex of the edge contained in the voxel.

It is possible to implement the operators β_i of the 3-map by traversing the crossings of each voxel as in a rPFC, and by moving along adjacent voxels (similarly to the function $convert$ in the rPFC). We formally describe the 3-rmap and its operations in this section.

3.1. Voxel grid

Let $[x_1,x_n]\times[y_1,y_n]\times[z_1,z_n]$ be a voxel grid. Each voxel contains six faces (or pixels), and each face contains four edges. We say that a 3-map is in *general position* with respect to a voxel grid if it satisfies the following conditions:

C1 A 0-/1-/2-cell does not intersect a face/edge/vertex of the voxel grid.
C2 A voxel does not contain any 1-cell in its interior.
C3 The intersection of a 2-cell with the boundary of a voxel is empty or connected.

Conditions **C1** and **C2** are plain generalizations of the conditions of a rPFC. Condition **C3** is novel and the analogous condition in 2D (the intersection of a face with the boundary of a pixel is empty or connected) is not necessary. Figure 4 shows an example in which condition **C3** is not satisfied.

Note that a voxel grid can be interpreted as a 3-map, which

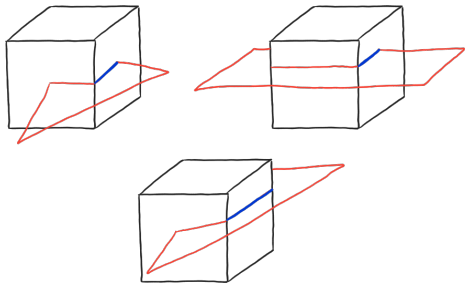

Figure 5: *Different intersections of a 2-cell with the boundary of a voxel. Top: crossings of type 0 and 1 (in blue). Bottom: a pair of crossings of type 2.*

we call *supporting 3-map* of the 3-rmap. Each dart is of the form (v', e', f', c'), where v' is a vertex incident to an edge e', which belongs to a pixel f' in the boundary of a voxel c'. It is easy to define the operators β_i of the supporting 3-map implicitly, so that they do not need to be build in memory.

3.2. Crossings and words

Intuitively, a crossing in a 3-rmap is the intersection of a 2-cell with a dart of the voxel grid. It is the intersection of a 2-cell f with the edge e' of a pixel f' of a voxel c' (that is, $e' < f' < c'$). We can classify a crossing in three classes:

- A crossing is of *type 0* if the 2-cell f has an incident 1-cell $e < f$ that intersects the pixel f' and an incident 0-cell v contained in the voxel c'. It is related to the crossings of the other 2-cells incident to e that intersect an edge of the same pixel f'.
- A crossing is of *type 1* if the 2-cell f has an incident 1-cell $e < f$ that intersects the pixel f', but it does not have an incident 0-cell v in the voxel c'. Similarly, it is related to the crossings of the other 2-cells incident to e that intersect an edge of the pixel f'.
- A crossing is of *type 2* if the 2-cell f does not have an incident 1-cell $e < f$ intersecting the pixel f'. It is related to the crossing of the 2-cell f with another edge of the pixel f'.

These three types of crossings are illustrated in Figure 5.

Like in the rPFC, we sort the crossings in the edges of each pixel following any fixed order on the edges of each face of the voxels. Then, the symbol of a crossing c depends on its type:

- Type 0. If it is the first crossing related to the 1-cell, its symbol is '`(`'. If it is the last, it is '`)`'. Otherwise, its is '`Y`'. If it is the only 2-cell incident to that 1-cell, its symbol is '`=`'.
- Type 1. If it is the first crossing related to the 1-cell, its symbol is '`{`'. If it is the last, it is '`}`'. Otherwise, its is '`Y`'. If it is the only 2-cell incident to that 1-cell, its symbol is '`#`'.
- Type 2. Its symbol is '`[`' if it is the first crossings, or '`]`' if it is the second one.

Hence, we build an *edge-word* by joining the symbols of the crossings in each edge and adding the symbol '+' at the end. Hence, we can identify a crossing by its edge e', its pixel f', its voxel c' and its position i in its edge-word.

A pixel-word is the concatenation of four edge-words. Note that, like in the rPFC, we can simplify the three closing symbols to a unique symbol.

3.3. Rasterized darts and operators β_i

Let $d = (v, e, f, c)$ be a dart of the 3-map. We define its associated *rasterized dart* as the pair

$$\rho(d) = (d', i) = (v', e', f', c'; i) \tag{1}$$

where d' is a dart of the supporting 3-map and i is an integer. Actually, c' is the voxel that contains the vertex v; f' is the pixel of c' that contains the intersection of e with the boundary of c'; and e' is the edge of f' that contains the intersection of f with the boundary of f'. The vertex of d' is derived from the other cells. Lastly, the index i is the *position* of the crossing of f in the edge-word of d'.

A rasterized dart is a crossing of type 0 in the 3-rmap. We use it to link the operators in the 3-rmap with the operators in the 3-map.

We now introduce the operators of the 3-rmap for traversing its rasterized darts. Let (d', i) be a crossing (of any type). The edge-word associated to d' is $w(d')$, and its length is $|w(d')|$. We define:

$$conv[3](d', i) = (\beta_3(d'), |w(d')| - i)$$
$$conv[2](d', i) = (\beta_2(d'), |w(d')| - i)$$

where β_3 and β_2 are the operators on the supporting 3-map (see Section 2.1 for their definition). $conv[3]$ maps a crossing to the crossing made by the same intersection, but in the opposite voxel. $conv[2]$ is like the function $convert$ in the rPFC, since it maps a crossing with its corresponding crossing in the opposite pixel.

Also, we define the functions $type[0]$ and $type[1]$ that return true iff a crossing is of type 0 or 1, respectively. This can be computed by examining its pixel-word.

We now introduce the following notation. Let (f_1, f_2, f_3) be a triplet of functions, then $F(x) := (f_1, f_2, f_3)(x)$ is computed as follows:

$y \leftarrow f_1(x)$;
`if`$(f_2(y))$ `return` $F(f_3(y))$;
`else` `return` y;

Now we can define the following operators

$$\beta_3[3] = (conv[3], type[0], \beta_2[2])$$
$$\beta_2[3] = \beta_1[2] \circ \beta_2[2] \circ \beta_3[3]$$
$$\beta_1[3] = \beta_2[2] \circ \beta_3[3]$$
$$\beta_0[3] = \beta_3[3] \circ \beta_2[2]$$
$$\beta_2[2] = (conv[2], type[1], next)$$
$$\beta_1[2] = next \circ \beta_2[2]$$
$$\beta_0[2] = \beta_2[2] \circ previous$$

The operators $\beta_i[2]$ correspond to applying β_i in the 2-map on the boundary of each voxel, and thus they generalize the operators of a rPFC. $conv[2]$ is used for moving between pixels of the same voxel. $\beta_2[2]$ is recursive to skip crossings of type 2 which do not correspond with darts in the 2-map, and it is used by $\beta_1[2]$ and $\beta_0[2]$.

All the operators $\beta_i[3]$ call the operators $\beta_i[2]$. $conv[3]$ allows us to move from one voxel to its adjacent voxel (along the face of the crossing). $\beta_3[3]$ implements the operator β_3 in the 3-map, and it skips the crossings of type 1.

We can prove that $\beta_i[3] \circ \rho = \rho \circ \beta_i$. This means that we can encode a 3-map (in general position) with a 3-rmap without loss of information.

3.4. A voxel grid is not enough

Given a 3-map, it is easy to find a voxel grid that satisfies conditions **C1** and **C2**. **C1** is usually satisfied, and if not, it suffices to translate the voxel grid by an arbitrarily small vector. For **C2**, it suffices to choose a sufficiently small grid size.

Surprisingly, **C3** is much harder to satisfy. Our experiments show that decreasing the grid size does not guarantee that we find a voxel grid in general position. Hence, it is necessary to consider a more elaborate supporting 3-map.

The most simple solution seems to be a hierarchical voxel grid, where each voxel, pixel and edge can be recursively subdivided into 8 voxels, 4 pixels and 2 edges, respectively. Starting from a fixed voxel grid, we can prove that by recursively subdividing the voxels that do not satisfy **C2** or **C3**, we eventually obtain a voxel grid in regular position in a finite number of steps. Then, we subdivide each face and edge so that the cells of the supporting 3-map are topologically coherent, that is they only intersect themselves along cells of lower dimension.

This has two impacts on the previous exposition of the 3-rmap in Section 3. First, the crossings (and thus the rasterized darts) must explicitly tell the sub-voxel, sub-pixel and sub-edge. This can be done using a Morton code. Second, the traversal of the supporting 3-map in $conv[3]$ and $conv[2]$ is a bit more intricate because we must know the structure of the hierarchical voxel grid.

In practice, we can avoid subdividing the edges by altering the definition of $conv[2]$. In a nutshell, we need to inspect the length of some neighboring edge-words to compute the index of the crossing.

A relevant advantage of using a hierarchical voxel grid is that we can adapt the detail of the 3-rmap to that of the object. This can avoid having many empty voxels, which saves memory space, and also having one edge intersecting many voxels, which reduces the execution time for traversing the rasterized darts.

4. Compact Implementations

In order to navigate through the rasterized darts of the 3-rmap, we need fast access to the symbol of any crossing and to the structure of the hierarchical voxel grid. Following the previous section, a crossing is of the form $(e, p, cp, v, cv; i)$ where $v \in \mathbb{N}^3$ are the coordinates of the (not-subdivided) voxel in the regular voxel grid, $cv \in [0,7]^*$ is the Morton code encoding the subdivided voxel, $p \in [0,5]$ is the face of the voxel, $cp \in [0,3]^*$ is the Morton code of the pixel, $e \in [0,3]$ is the edge of the pixel and i is the position of the crossing in its edge-word.

We recall that the voxel-words are made of 8 different symbols (cf. Section 3.2), so we can encode each symbol with 3 bits.

We use two compact constant-access containers. The fixed-length container is a vector of sequences of bits, which is encoded in an array of bits plus its number of elements and the size of the entries. The variable-length container needs an extra vector with the position of each entry.

In the rest of this section we describe four strategies for compact representations of the 3-rmap with different trade-offs between memory space and time complexity for queries in its structure, and we compare them with two 3D models.

4.1. V1: Explicit structure

According to the structure of a crossing in a hierarchical voxel grid, it is clear that the edge-words can be stored in a 3D array of octrees of arrays of quadtrees of arrays of words.

A quadtree (or octree) can be encoded in a vector in the following way: each entry corresponds to a node of the quadtree following the breadth-first order. It contains a bit to tell if it is an internal node or a leaf, and the index in the vector of its first child (divided by four), or the key of the leaf.

Then, all the edge-words can be encoded in a single fixed-length vector. Note that the size of this structure is influenced by the number of edge-words and the length of the longest edge-word.

4.2. V2: Octree and dictionary of voxel-words

We can encode a quadtree of pixel-words in a single *super-pixel-word*: traverse the subdivided pixels following the

depth-first order and put a special symbol 'p' for the internal nodes, or a pixel-word for the leaves. Then, we define a *voxel-word* as the concatenation of its six super-pixel-words.

A 3-rmap contains as many voxel-words as voxels there are in the hierarchical voxel grid, and many of them usually coincide. Our second implementation consists of a dictionary of voxel-words and an octree. All the different voxel-words are stored in a vector. To save up memory space, we encode them with a Huffman code [CLRS09, §16.3]. Infrequent voxel-words have longer codes, so we implement the dictionary of voxel-words in a variable-length vector. We implement the octree in a fixed-length vector, where each leaf contains the index of its voxel-word.

This implementation is more compact than V1 if there are many repeated voxel-words, whereas the access to a given edge-word has linear-time complexity in the size of its voxel-word.

4.3. V3: Octree and dictionary of voxel-words with encoded pixel-words

The previous implementation can be extended a bit further. The number of different pixel-words in the voxel-words of the dictionary of V2 is relatively small, and the pixel-words appear with varying frequencies. Thus, we can encode the pixel-words (and the one-symbol word 'p') with a Huffman code to save memory space for the dictionary of voxel-words.

The binary tree to decode the Huffman code is stored in a variable-length vector. Then, each voxel-word is a sequence of bits (the concatenation of the codes of its pixel-words). The dictionary of voxel-words is implemented in a variable-length vector.

This implementation is more compact than V2 if the number of different pixel-words is small enough compared to their average length.

4.4. V4: Only one word

We can define *super-voxel-words* in a similar way as for the super-pixel-words, by adding yet another symbol 'v' for subdivided voxels. Hence, we can write all the edge-words in a single word (the concatenation of all the super-voxel-words in the raster order) and compress it with a Huffman code.

This implementation can be advantageous if the entropy of the word is small enough. However, the access to the symbol of a crossing is linear in the size of the word.

4.5. Memory comparison

We illustrate the storage cost of each implementation by estimating them for the two volumic meshes in Figure 6: HOUSE and BUILDING. The former has 384 vertices, 1470 edges, 112 faces and 47 volumes. The latter is significantly larger, since it has 2628 vertices, 10226 edges, 7814 faces and 298 volumes. The results are shown in Table 1. An early conclusion is that the most compact implementation is V3. We now discuss our results in more detail.

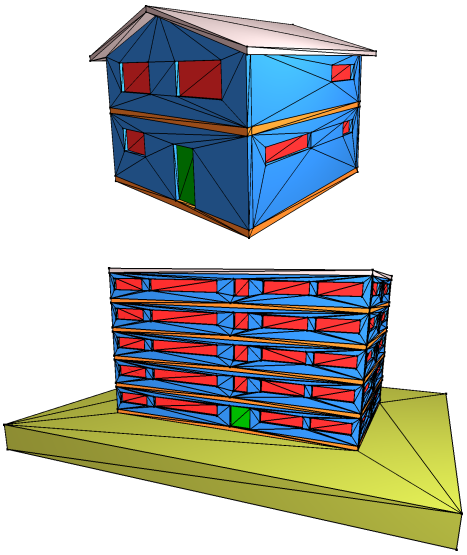

Figure 6: *The two volumic meshes considered for the experiments:* HOUSE *and* BUILDING.

	V1	V2	V3	V4
HOUSE	24444	779	691	1423
BUILDING	750791	21324	20499	36263

Table 1: *Memory space (in kB) of each implementation for the two models.*

The 3-rmap of HOUSE was computed from an initial voxel grid with only one voxel. Its edge-words are very short, having at most three symbols.

The fixed-length vector of V1 has 8.5 million entries, with 28% of them encoding indices of the vector and the rest, edge-words. Hence, we need more space for storing an index (24 bits) than an edge-word (9 bits), so we *waste* 15 bits of memory for each edge-word.

The fixed-length vector of V2 that encodes the octree of voxels only contains 290865 entries (29 times less than V1), while the dictionary has 6992 voxel-words. There are 254507 voxel-words in the 3-rmap (with repetitions), with lengths from 24 to 1558 symbols. There is still a small gap between the indices for the internal nodes of the octree (16 bits) and the indices for the dictionary (13 bits). The size of the dictionary is 161.7 kB, that is 20.7% of the total. Note that other (less compact) possibilities for encoding the dictionary of voxel-words are: fixed-length vector without coding (6583 kB), variable-length vector without coding (435 kB) and fixed-length vector with Huffman coding (1896 kB).

The dictionary of voxel-words together with the binary

tree for the Huffman code in V3 occupy 72.2 kB + 1 kB (55% less then V2). There are 92 different pixel-words with lengths ranging from 4 to 10 symbols.

The word in V4 has 8.9 million symbols with an entropy of 0.79 bits/symbol. It occupies 1423 kB (against 4455 kB without Huffman encoding).

The results on BUILDING are similar. The only difference with respect to HOUSE is that the dictionary of voxel-words in V2 involves only 6.6% of the total memory space, and thus the gain of V3 is less important.

V1 offers constant-time access to the symbol of any crossing. On the other hand, V4 contains the same information without the explicit structure of the hierarchical voxel grid, so it has linear-time access to the symbols. Thus, both versions can be seen as the extrema of the trade-off between memory space and time for a 3-rmap. Interestingly, both V2 and V3 propose an intermediate time complexity, while their memory space is inferior to that of V4 thanks to the indexing of the voxel-words.

5. Conclusion

This work shows that it is possible to extend the rasterized planar face complex data structure to 3D, and probably to higher dimensions. The immediate advantage of this representation of subdivided objects is that it partitions the object into blocks of information that can be streamed during its traversal.

The two differences with respect to the rasterized planar face complex are:

1. We need two more symbols for the edge-words.
2. We need to work on a hierarchical voxel grid, where each voxel can be recursively divided into 8 sub-voxels, and each pixel (voxel face) can be recursively divided into 4 sub-pixels.

In a hierarchical voxel grid, we have to explicitly represent its structure to apply the functions $conv[3]$ and $conv[2]$ for navigating through the voxels and pixels. However, this overhead in memory is usually canceled out by the reduced number of empty voxel-words.

We plan to reduce the subdivision of the voxels by adapting the refinement scheme to each case. Instead of always dividing a voxel into eight identical sub-voxels, we can set different sizes to avoid later subdivisions in its sub-voxels.

We also intend to validate our approach with a larger data set and compare the time complexity of the different implementations. A later perspective is to generalize this work to any dimension.

Références

[Bri89] BRISSON E.: Representing geometric structures in d dimensions: topology and order. In *Proc. of ACM Symposium Computational Geometry* (Saarbrücken, Germany, June 1989), pp. 218–227.

[CLRS09] CORMEN T. H., LEISERSON C. E., RIVEST R. L., STEIN C.: *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009.

[dB00] DE BERG M.: *Computational geometry: algorithms and applications, 2nd Edition*. Springer, 2000.

[DL87] DOBKIN D., LASZLO M.: Primitives for the manipulation of three-dimensional subdivisions. In *Proc. of Symposium on Computational Geometry* (Waterloo, Canada, June 1987), pp. 86–99.

[DL14] DAMIAND G., LIENHARDT P.: *Combinatorial Maps: Efficient Data Structures for Computer Graphics and Image Processing*. A K Peters/CRC Press, septembre 2014.

[DR17] DAMIAND G., ROSSIGNAC J.: Rasterized planar face complex. *Computer-Aided Design. Vol. 90* (2017), 146–156.

[DRE*15] DYEDOV V., RAY N., EINSTEIN D. R., JIAO X., TAUTGES T. J.: AHF: array-based half-facet data structure for mixed-dimensional and non-manifold meshes. *Eng. Comput. (Lond.). Vol. 31*, Num. 3 (2015), 389–404.

[GS85] GUIBAS L. J., STOLFI J.: Primitives for the manipulation of general subdivisions and computation of Voronoi diagrams. *ACM Trans. Graph.. Vol. 4*, Num. 2 (1985), 74–123.

[Män88] MÄNTYLÄ M.: *An Introduction to Solid Modeling*. Computer Science Press, 1988.

[Ros01] ROSSIGNAC J.: 3D compression made simple: Edgebreaker with Zip&Wrap on a corner-table. In *2001 International Conference on Shape Modeling and Applications (SMI 2001), 7-11 May 2001, Genoa, Italy* (2001), p. 278.

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

[Wei88] WEILER K.: The radial edge structure: a topological representation for non-manifold geometric boundary modeling. In *Geometric Modeling for CAD Applications*. Elsevier Science, 1988, pp. 217–254.