BMT: A Generic Programming Approach to Multiresolution Spatial Decompositions

Vinícius Mello[1], Luiz Velho[1], Paulo Roma Cavalcanti[2] and Cláudio Silva[3]

[1]IMPA – Instituto de Matemática Pura e Aplicada
{vinicius,lvelho}@visgraf.impa.br

[2]Federal University of Rio de Janeiro - UFRJ
roma@lcg.ufrj.br

[3]AT&T Labs
csilva@research.att.com

Abstract

We present a generic programming approach to the implementation of multiresolution spatial decompositions. From a set of simple and necessary requirements, we arrive at the Binary Multitriangulation (BMT) concept. We also describe a data structure that models the BMT concept in its full generality.

1 Introduction

Generic programming was born from the observation that most algorithms rely on a few basic semantic assumptions about the data structures, and not on any particular implementation of these structures. Given a problem, the generic programming basic task is to isolate these essential concepts, framing them in a well defined interface where semantic requirements and computational complexity guarantees are clearly posed. The algorithms that comply with that interface are free from idiosyncrasies of data structures, which can be changed or even replaced by procedural schemes. The great sucess of C++ Standard Template Library is the main proof on behalf of that methodology [14].

The main contribution of this paper is to show how generic programming techniques can be used to build a computational framework to deal with multiresolution spatial decompositions. We have studied from combinatorial topology classics like [1] to modern works on multiresolution modeling [8] in order to identify the meaningful concepts. As a result, we arrive at a new concept called Binary Multitriangulation (BMT) that is a particular case of the Multiresolution Simplicial Model (MSM) described in [8], but more manageable and closer in spirit to well stablished procedures of combinatorial topology. The BMT concept can also be regarded as a 3-dimensional extension of variable resolution structures like [16].

Currently, generic programming methodology is being used as design philosophy of many libraries in several areas: computational geometry (CGAL [7]), combinatorics (BGL [12]) and scientific computing (MTL [13]), for instance. An interesting aspect of generic programming, illustrated in section 5, is orthogonality, that is, the software components, even from different libraries, can be freely combined. Other aspect is the omnipresence of C++ language, the most fitted language to generic programming (specially after the recent ISO/ANSI standardization). This fact justifies the extensive use of C++ code in this paper, but we must emphasize that there is no intrinsic dependence to specific programming languages in the concepts we describe.

The paper is organized as follows. In section 2, we set the context where multiresolution spatial decompositions are needed and discuss informally some of their advantages. Section 3 is dedicated to the detailed examination of the concepts we isolate, in increasing order of complexity. In section 4, we describe data structures that models the previously defined concepts. Finally, section 5 contains concluding remarks and indications of future works, including applications.

2 Background

If multiresolution methods are signficant in the processing of triangle meshes [6, 5], they are indispensable in the case of tetrahedra meshes, or spatial decompositions, as we prefer to call it, since the complexity of the mesh increases with the power of its dimension. Therefore, most applications dealing with three-dimensional data, like scientific visualization, medical imaging, geoprocessing etc., will benefit of techniques that allow the user to extract from the original data an equivalent representation, in a sense will be made precise subsequently, but in a resolution more adequate to the task at hand. Because the user often doesn't know *a priori* the desired resolution, the solution is to store a good set of possible resolutions and to give him the ability of browsing between them. That is the essence of the multiresolution methods.

A cost-benefit analysis of multiresolution methods with respect to the current technology is presented in [2]. The main conclusion of this analysis is that, in the case of direct volume rendering applications (DVR), the graphics constraints are stronger than memory constraints. In fact, one can store a mesh three orders of magnitude larger than that one can visualize with DVR. It is reasonable to assume that similar conclusions can be extended to other applications dealing with volume data. Thus, the extra cost to store a multiresolution mesh structure is compensated by the flexibility to choose the most adequate resolution.

It remains to define more precisely what is a tetrahedral mesh. Although, the basic intuitive concept is clear, it is not so clear that some properties of a mesh are independent of the geometry of its constituent tetrahedra or, more specifically, of their spatial embedding. The area of mathematics that studies such properties is called *combinatorial topology*. A *manifold* is a well known mathematical object and it is very useful in combinatorial topology. A combinatorial manifold is characterized by a certain uniformity in the way its parts are glued. Despite of the fact that in some applications it is necessary to consider "non-manifold" structures, the concept of a manifold is sufficient for most applications.

In the next session, we will describe a series of concepts progressively more complex, until we arrive at a definition of a combinatorial manifold. Each concept will be followed by an API which defines the operations required to work with the concept. We will postpone the introduction of geometric concepts as much as possible, in order to clearly isolate the topological properties.

However, geometric concepts, such as volume, area, aspect ratio, etc., are of fundamental importance in applications and are deeply related with the mechanisms that are employed to select a particular mesh resolution from a multiresolution structure. In general, we deal with functions whose domain is the combinatorial manifold, and we would like to have a more refined mesh in regions where these functions exhibit high variations. This property is called *adaptivity*. We intend to discuss techniques for generation and processing of multiresolution adaptive tetrahedral meshes in a future paper.

3 Concepts

In this section, we adopt the following strategy in describing the concepts: we will define the mathematical objects involved, the name and type signature of the requirements, and let the semantics be derived from them and from hints in the text body. Auxiliary tables containing associated types and notations fulfill the description. Concerning to type signatures, each concept has a *trait class* where all type information is encoded. Trait classes are essentially a mechanism to ensure algorithm independence of data structure implementation [15].

Some definitions bellow differs from the usual (see [3], for instance), but this happens because we have choosen equivalent definitions easily translatable to algorithm requirements.

description	type
Vertex descriptor	as3c_traits<T>::vertex_descriptor
Edge descriptor	as3c_traits<T>::edge_descriptor
Face descriptor	as3c_traits<T>::face_descriptor
Simplex descriptor	as3c_traits<T>::simplex_descriptor
Complex vertices iterator	as3c_traits<T>::vertex_iterator
Complex edges iterator	as3c_traits<T>::edge_iterator
Complex faces iterator	as3c_traits<T>::face_iterator
Complex simplices iterator	as3c_traits<T>::simplex_iterator

Table 1: Associated types of an AS3C.

3.1 Abstract Simplicial 3-Complex

Definition 1. *Given a finite set V, called* vertex set*, an* abstract simplicial complex *on V is a set K of subsets of V verifying the following properties:*

1. *For each $\nu \in V$, $\{\nu\} \in K$;*
2. *If $\sigma \in K$ and $\phi \subset \sigma$, them $\phi \in K$. An element σ of K is called* simplex *and the subsets of σ are called* faces*;*
3. *There is a total ordering on the vertices of each simplex of K such that the ordering on the vertices on any face of a simplex σ is the ordering induced from the ordering on the vertices of σ.*

If $n+1$ is the cardinality of a simplex $\sigma \in K$, we say that σ is a n-simplex and K is an *abstract simplicial n-complex* if the largest simplex of K is a n-simplex. We will use the same name simplex to mean the subcomplex of K formed by the faces of a simplex σ, and we define $\partial\sigma = \{\tau \in K : \tau \subsetneq \sigma\}$, that is, the *boundary* of σ.

We are interested here in abstract simplicial 3-complexes, AS3C for short. In this case, we adopt the terminology *vertex*, *edge*, *face* and *simplex* for 0, 1, 2 and 3-simplex, respectively.

Item 3 from definition 1 has a twofold propose: it provides a "canonical form" for each simplex and enables us to define the *face operator* d_i, that assigns for each simplex σ the face of σ obtained by removing the i-th vertex. The face operator satisfies

$$d_i d_j = d_{j-1} d_i, \text{ if } i < j. \tag{1}$$

The existence of operators satisfying (1) is sufficient to recover all relations between the faces of a simplex[1]. Figure 1 shows all that relations. We exploit this fact to define a minimum set of requirements on a AS3C (see table 3). Program 1 shows how to get the i-th vertex of a simplex and program 2 implements a vertex-simplex membership test. Of course, we can specialize both programs if more information about the underlying data structure is known.

3.2 Abstract 3-Manifold

In principle, the requirements on a AS3C are enough to answer incidence queries like "get all faces meeting an edge" or "get all edges meeting a vertex". But, in many cases, we have more information about the

[1]Face operators appear in algebraic topology in the definition of simplicial sets [10].

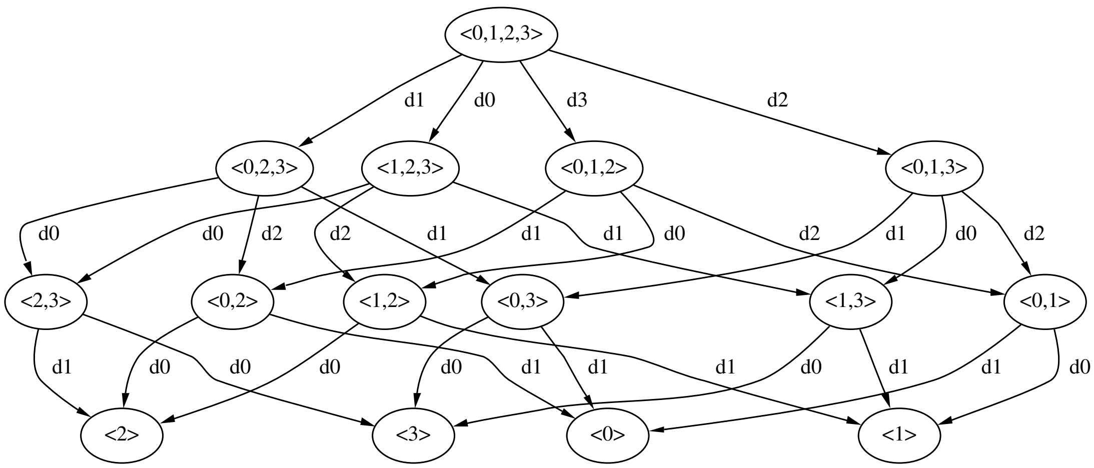

Figure 1: Face operator graph. The nodes are the simplex's faces and the edges are the face operators d_i.

symbol	definition
vertex	typedef as3c_traits<T>::vertex_descriptor vertex;
v	a object of type vertex
edge	typedef as3c_traits<T>::edge_descriptor edge;
e	a object of type edge
face	typedef as3c_traits<T>::face_descriptor face;
f	a object of type face
simplex	typedef as3c_traits<T>::simlpex_descriptor simplex;
s	a object of type simplex
vi	typedef as3c_traits<T>::vertex_iterator vi;
ei	typedef as3c_traits<T>::edge_iterator ei;
fi	typedef as3c_traits<T>::face_iterator fi;
si	typedef as3c_traits<T>::simplex_iterator si;

Table 2: AS3C related notation.

expression	return type
empty_vertex(t)	vertex
empty_edge(t)	edge
empty_face(t)	face
empty_simplex(t)	simplex
face_op(t, s, i)	face
face_op(t, f, i)	edge
face_op(t, e, i)	vertex
vertices(t)	pair<vi, vi>
edges(t)	pair<ei, ei>
faces(t)	pair<fi, fi>
simplices(t)	pair<si, si>

Table 3: Requirements of an AS3C. Some remarks about the notation: `t` is a object which type models a AS3C; `empty_vertex(t)` returns a null vertex descriptor; `vi` is the type of a iterator which traverses the vertex container; `vertices(t)` return a pair of iterators: the first points to the first vertex and the second is a "past-the-end" iterator. The other operators work analogously.

```
template <typename T> as3c_traits<T>::vertex_descriptor ith_vertex(T t, as3c_traits<T>::simplex_descriptor s, int i) {
  if(i<2) {
    if(i==0) return face_op(t, face_op(t, face_op(t, s, 1), 1), 1);
    else return face_op(t, face_op(t, face_op(t, s, 0), 1), 1);
  } else {
    if(i==2) return face_op(t, face_op(t, face_op(t, s, 0), 0), 1);
    else return face_op(t, face_op(t, face_op(t, s, 0), 0), 0);
  }
}
```

Program 1: i-th vertex of a simplex. Note how the trait class `as3c_traits` isolates the algorithm from the data structure implementation.

local structure of the complex in each vertex. That information can be used to speed-up those queries. To describe precisely that local structure, we need some definitions.

Two simplices σ_1, σ_2 are *independent* if $\sigma_1 \cap \sigma_2 = \emptyset$. The *join* $\sigma_1 \star \sigma_2$ of independent simplices σ_1, σ_2 is the set $\sigma_1 \cup \sigma_2$. The join of complexes K and L, written $K \star L$, is $\{\sigma \star \tau : \sigma \in K, \tau \in L\}$. The *link* of simplex $\sigma \in K$, denoted $\operatorname{link}(\sigma, K)$, is defined by

$$\operatorname{link}(\sigma, K) = \{\tau \in K : \sigma \star \tau \in K\}.$$

And finally, the *star* of σ in K, $\operatorname{star}(\sigma, K)$, is the join $\sigma \star \operatorname{link}(\sigma, K)$.

The link and star operators provides a combinatorial description of a neighborhood of a simplex. We can use them also to define certain changes in a complex, but care must be taken to not modify essentially ("topologically") that neighborhood.

The *stellar moves* are a such change. Indeed, many concepts of combinatorial topology are founded on stellar moves [9]. Let K be a complex on the vertex set V, K' a complex on V', σ a simplex in K and ν a vertex in V'. The operation that changes K into K' by removing $\operatorname{star}(\sigma, K)$ and replacing it with $\nu \star \partial\sigma \star \operatorname{link}(\sigma, K)$ is called a *stellar subdivision* and is written $K' = (\sigma, \nu)K$. The inverse operation $(\sigma, \nu)^{-1}$ that changes K' into K is called a *stellar weld*.

Two complexes are *stellar equivalent* if they are related by a sequence of stellar moves. A (abstract)

```
template <typename T>
bool in(T t, as3c_traits<T>::simplex_descriptor s,
  as3c_traits<T>::vertex_descriptor v) {
  for(int i=0; i<4; ++i) if(ith_vertex(t, s, i)==v) return true;
  return false;
}
```

Program 2: Vertex-simplex membership test.

description	type
Incident faces iterator	a3m_traits<T>::radial_face_iterator
Incident simplices iterator	a3m_traits<T>::radial_simplex_iterator

Table 4: Associated types of an A3M.

n-*ball* is a complex stellar equivalent to a n-simplex and a (abstract) n-*sphere* is a complex stellar equivalent to the boundary of a $(n+1)$-simplex.

We can now define a special kind of abstract simplicial complex that has nice local properties.

Definition 2. *An* abstract n-manifold M *is an abstract simplicial* n*-complex such that for each vertex* $\nu \in M$*,* $\text{link}(\nu, M)$ *is a* $(n-1)$*-ball or a* $(n-1)$*-sphere.*

The *boundary of* M, denoted by ∂M, is the subcomplex $\partial M = \{\sigma \in M : \text{link}(\sigma, M) \text{ is a ball}\}$. One can proof that ∂M is a $(n-1)$-manifold.

Many properties follows from definition 2. In our particular case, we want properties that help us to speed-up local queries over abstract 3-manifolds (A3M). Each boundary face of a 3-manifold, for example, is incident to a unique simplex and internal faces (faces not on boundary) are shared by exactly two simplices. Therefore, we can require operators that, in constant time, retrieve all simplices meeting at a face.

Table 6 lists a set of additional requirements to the AS3M ones we judge sufficient to formalize the A3M concept. With that additional requirements, we can implement program 3 which, given an internal face f and an incident simplex s, returns the simplex that shares f with s. That modest operation is the key component of a *radial iterator*, that is, an iterator that traverses all faces or simplices meeting an edge. Radial iterators are used in algorithms that compute the star of vertices and edges, for instance, and are reminiscent of the Weiler's radial edge structure (RED) [17]. Again, nothing prevents users from implementing *ad hoc* iterators, perhaps based on some reliable implementation of the facet-edge structure of Dobkin and Laszlo [4].

3.3 Oriented Abstract 3-Manifold

Orientation is another notion we want capture. Since orientation can be defined in a purely combinatorial way, without reference to geometrical concepts, we choose to place the *oriented abstract* 3*-manifold* concept as a refinement of abstract 3-manifold.

symbol	definition
rfi	typedef a3m_traits<T>::radial_face_iterator rfi;
rsi	typedef a3m_traits<T>::radial_simplex_iterator rsi;

Table 5: A3M related notation.

Refinement of abstract simplicial 3-complex	
expression	return type
on_boundary(t, v)	bool
on_boundary(t, e)	bool
on_boundary(t, f)	bool
incident_simplex(t, f)	simplex
incident_simplices(t, f)	pair<simplex, simplex>
a_incident_face(t, e)	face
boundary_faces(t, e)	pair<face, face>
a_incident_edge(t, v)	edge
radial_simplices(t, e)	pair<rsi, rsi>
radial_faces(t, e)	pair<rfi, rfi>

Table 6: Requirements of an A3M. Type t models a A3M. Types rsi and rfi model a radial simplex iterator and radial face iterator, respectively. Some pre-conditions must hold: boundary_faces(t, e) can be used only if on_boundary(t, e)==true, for instance.

```
template <typename T> as3c_traits<T>::simplex_descriptor
opposite_simplex(T t, as3c_traits<T>::simplex_descriptor s, as3c_traits<T>::face_descriptor f) {
  as3c_traits<T>::simplex_descriptor s1, s2;
  tie(s1, s2)=incident_simplices(t, f);
  if(s1==s) return s2 else return s1;
}
```

Program 3: Opposite simplex. The tie function above is just a compact way of assign a pair of values to two variables.

An *orientation* on a n-manifold M is a function s that assigns for each n-simplex $\sigma \in M$, an integer in the set $\{+1, -1\}$. The choice of orientation in σ induces an orientation in its faces in the following way:

$$s(d_i(\sigma)) = (-1)^i s(\sigma), i = 0, \dots, n. \tag{2}$$

An orientation is *coherent* if *contiguous* n-simplices, i.e., simplices sharing an $(n-1)$-simplex, induces opposites orientations in its common face, that is,

$$d_i(\sigma_1) = d_j(\sigma_2) \Rightarrow s(d_i(\sigma_1)) = -s(d_j(\sigma_2)),$$

where σ_1 and σ_2 are n-simplices in M. Now, we can define another basic object.

Definition 3. *An* oriented abstract n-manifold *is an abstract n-manifold plus a coherent orientation.*

In the 3-dimensional case, the additional requirement on a A3M is just the operator simplex_orientation that takes a manifold and a simplex and returns an int in the set $\{-1, 1\}$. Program 4 is the obvious implementation of the equation 2.

3.4 3-Polyhedron and Combinatorial 3-Manifold

Until now, we discussed only combinatorial concepts. Let's introduce the geometrical counterpart of the previously defined concepts.

We call an *euclidean embedding* of an abstract simplicial complex K a function g from the vertex set V to an euclidean space E^m that maps a vertex $\nu \in V$ to a euclidean point $g(\nu) \in E^m$, such that $g(\sigma)$ is a set

Refinement of abstract 3-manifold	
expression	return type
simplex_orientation(t, s)	int

Table 7: Requirements of an OA3M.

```
template <typename T>
int face_orientation(T t, as3c_traits<T>::simplex_descriptor s, as3c_traits<T>::face_descriptor f) {
  int o=simplex_orientation(t, s);
  for(int i=0; i<4; ++i, o*=-1) if(ith_face(t, s, i)==f) return o;
}
```

Program 4: Induced orientation of the faces.

in general position in E^m, for all $\sigma \in K$. A subset $\mathcal{P}$ of E^m is a *geometric realization* of K if there is an embedding g satisfying

$$x \in \mathcal{P} \Leftrightarrow x \in \text{ConvHull}(g(\sigma)), \text{ for some } \sigma \in K.$$

Below, we define the geometrical objects corresponding to abstract simplicial n-complex and abstract n-manifold.

Definition 4. *A n-*polyhedron *is a set $\mathcal{P} \subset E^m$ for which exists an abstract simplicial n-complex K and an euclidean embedding g such that $\mathcal{P} = |K|_g$.*

Definition 5. *A* combinatorial n-manifold *is a set $\mathcal{M} \subset E^m$ for which exists an abstract n-manifold M and an euclidean embedding g such that $\mathcal{M} = |M|_g$.*

From the computational side, a polyhedrom concept (Poly3) is just a refinement of AS3C with an additional requirement euclidean_point that takes a manifold and a vertex and return an euclidean point, see table 10. A combinatorial 3-manifold (C3M) is a Poly3 plus the requirements of an A3M.

3.5 Binary Multitriangulation

Now, we'll investigate the interplay between combinatorial and geometrical concepts related to subdivision process and how this leads naturaly to the concept of binary multitriangulations.

A polyhedron $\mathcal{P}' = |K'|_g$ is a *subdivision* of the polyhedron $\mathcal{P} = |K|_h$, denoted by $\mathcal{P}' < \mathcal{P}$, if $\mathcal{P}' = \mathcal{P}$ and for each $\sigma' \in K'$ exists a $\sigma \in K$ such that

$$\text{ConvHull}(g(\sigma')) \subset \text{ConvHull}(h(\sigma)).$$

The above definition uses geometrical concepts like euclidean embeddings. Therefore, we can not assert *a priori* anything about how the complexes K and K' are related. However, a theorem of Newman, presented

description	type
Point type	poly3_traits<T>::point_type

Table 8: Associated types of a Poly3.

symbol	definition
point	typedef poly3_traits<T>::point_type point;

Table 9: Poly3 related notation.

Refinement of abstract simplicial 3-complex	
expression	return type
euclidean_point(t, v)	point

Table 10: Requirements of a Poly3.

in modern form in [9], shows that $P' < P$ if, and only if, K' is stellar equivalent to K. Moreover, the stellar equivalence can be choosed in such a way that only stellar moves on 1-simplices ("edges") are used.

There is a good reason to restrict the stellar moves to moves on edges. Whenever a stellar subdivision happens in an edge ε, all simplices containing ε are splitted in two. Accordingly, a sequence of stellar subdivision induces a binary tree structure in the simplices. And binary trees often leads to simpler algorithms.

In order to define the binary multitriangulation concept (BMT), we need some auxiliary definitions. We follow closely the definitions in [8]. A *partially ordered set* (poset) $(C, <)$ is a set C with a antisymmetric and transitive relation $<$ defined on its elements. Given $c, c' \in C$, notation $c \prec c'$ means $c < c'$ and there in no $c'' \in C$ such that $c < c'' < c'$. An element $c \in C$, such that for all $c' \in C$, $c \leq c'$, is called a *minimal* element in C. If there is a unique minimal element $c \in C$, then c is called the *minimum* of C. Analogously are defined *maximal* and *maximum* elements.

Definition 6. *A* binary multitriangulation *is a poset* $(\mathcal{T}, <)$*, where* $\mathcal{T}$ *is a finite set of abstract* 3*-manifolds (named* triangulations*) and the order* $<$ *satisfies:*

1. *There is maximum and minimum abstract* 3*-manifolds in* $\mathcal{T}$*, called* base triangulation *and* full triangulation*, respectively;*

2. $M' \prec M$ *if, and only if,* $M' = (\varepsilon, \nu)M$*, for some edge* $\varepsilon \in M$*.*

Property 1 says, in fact, that a BMT is a *lattice*. Other fact which follows from the definition is that every two triangulations in $\mathcal{T}$ are stellar equivalent. As usual, a BMT can be thought as a directed acyclic graph (DAG), with one drain and one source, whose arrows are labeled with stellar subdivisions on edges. From an algorithimic perspective, the key idea is to use the above mentioned binary tree structure in the simplices to encode the DAG.

To describe the requirements on a BMT, we need to do a little digression about *state changes* in a data structure. The formerly defined requirements, like incident_simplices, are deterministic functions without side effects, at least from the user viewpoint. In other words, incident_simplices must return the same value in sucessive invocations. The situation changes in the BMT case, because we want to be able to move from a triangulation in $\mathcal{T}$ to another. We can regard this move as a state change in the underlying data structure modeling a BMT. The point is that, between state changes, the functions like incident_simplices behaves deterministically.

The BMT requirements in table 11 are divided in two groups: operators that changes the state (subdivide, weld and base_triangulation) and the others. The operator base_triangulation set the current triangulation in $\mathcal{T}$ to the base triangulation, while subdivide(t, e) applies a stellar subdivision to the edge e and weld(t, v) applies a stellar weld "removing" the vertex v. The predicate is_current is usefull to check if a simplex belongs to the current triangulation.

We must remark that operators subdivide and weld implements just “local” transitions in the DAG, that is, if $\mathcal{T}$ and $\mathcal{T}'$ are the triangulations before and after subdivide be called, respectively, then $\mathcal{T}' \prec \mathcal{T}$. Program 5 illustrates how subdivided_edge can be used to achieve non-local transitions.

Refinement of abstract 3-manifold	
expression	return type
was_subdivided(t, e)	bool
subdivide(t, e)	void
in_base_triangulation(t, v)	bool
weld(t, v)	void
base_triangulation(t)	void
has_children(t, s)	bool
children(t, s)	pair<simplex, simplex>
has_parent(t, s)	bool
parent(t, s)	simplex
subdivided_edge(t, s)	edge
welded_vertex(t, s)	vertex
is_current(t, v)	bool
is_current(t, e)	bool
is_current(t, f)	bool
is_current(t, s)	bool

Table 11: Requirements of a BMT. Type `t` models a BMT. The binary tree structure in the simplices can be traversed with children and parent.

```
template <typename T> void non_local_subdivide(T t, as3c_traits<T>::edge_descriptor e) {
  a3m_traits<T>::radial_simplex_iterator i, end;
  set<as3c_traits<T>::edge_descriptor> edges;
  for(tie(i, end)=radial_simplices(t, e); i!=end; ++i) {
    as3c_traits<T>::edge_descriptor se=subdivided_edge(t, *i);
    if(se!=e) edges.insert(se);
  }
  set<as3c_traits<T>::edge_descriptor>::iterator j;
  for(j=edges.begin(); j!=edges.end(); ++j) {
    non_local_subdivide(t, *j);
  }
  subdivide(t, e);
}
```

Program 5: Non-local subdivide. This program also ilustrates how the encoding of traversal capabilities in radial iterators makes the algorithm more generic and readable at no extra cost.

4 Models

In this section, we present data structures which are models of the concepts defined above, in the sense that they fill all necessary requirements. We have absolutely no pretension of describing “the best” data structure, for the reason that we think that data structures are somehow application dependent. But, if we did a good analysis in the previous section, most algorithms can be used in different applications without change.

Figure 2 resumes the prototypical data structure which models a BMT. Most operators are easily inferred by inspection. Stripping out some data, we obtain models to simpler concepts like AS3C and A3M. It remains to clarify certain points:

- We employ the classical "uses" device to represent the property that one face is common to two simplices at most;
- The fields fu_ind0 and fu_ind1 are indices to the subdivided faces. The operator subdivided_edge use them to return the subdivided edge;
- The field f_pair in struct edge stores the boundary faces incident to a boundary edge, or stores a face incident to a internal edge;
- A face f is on boundary if, and only if, f->fu_ptr->fu_mate_ptr is null, and a edge e is on boundary if, and only if, e->f_pair.first!=e->f_pair.second.
- A face is current if, and only if, its incident simplices are. And a edge is current if, and only if, one of its incident faces is.
- A ordering in adopted in the vertices of the simplex in such a way that the welded vertex is always the last vertex.

We note that this data structure is too general. Once more, the application guides the real implementation. In dealing with regular spatial decompositions, for example, most of work can be done procedurally. Even out-of-core techniques (e.g., [5]) can be implemented without changing the interface.

5 Conclusion

We have presented a generic programming framework for multiresolution spatial decompositions which was formulated through a rigorous mathematical analysis of the concepts involved. Indeed, our current implementation contains numerous generic algorithms for the extraction of topological information, as well as, non-generic functions to build a multiresolution mesh and execute other operations such as input/output. It is certainly possible to employ generic programming techniques in the creation of multiresolution meshes. Nonetheless, the problem is more involved and we plan to consider it in a future paper.

We are working on an application for direct volume rendering. We plan to use the multiresolution features of the BMT concept to achieve better rendering performance. At the moment, we have implemented the basic DVR pipeline as described in [18] and [11]. A very nice feature of our implementation is the way we compute the visibility order of the tetrahedra. We develop a *visibility graph* concept that is an adaptor over the C3M concept, i.e., it provides the appropriate graph interface to a combinatorial manifold. So, we can use all graph machinery built in the Boost Graph Library [12], for example, to solve the visibility problem. More specifically, in the simpler regular convex case, the visibility is computed by a single call to the BGL function topological_sort.

We think that our work can be classified in the confluence of two new trends in graphics. On one hand, our concern to clearly separate geometric and topological concepts, emphasizing the later ones, brings us closer to *Computational Topology* as posed in [3]. This new branch is as promising today as Computational Geometry was thirty years ago. On the other hand, generic programming is a powerful methodology for computer programming, which holds the promise to complete, specially in relation to algorithm abstraction, the revolution started twenty years ago by object-oriented programming. Our hope is that this work will become the basis of a library called CTAL, that is, Computational Topology Algorithm Library.

(a)
```
struct bmt {
  list<simplex *> s_list;
  list<face *> f_list;
  list<edge *> e_list;
  list<vertex *> v_list;
};
```

(b)
```
struct face {
  struct {
    unsigned id : 32;
  } bits;
  faceuse * fu_ptr;
  edge * e_vector[3];
};
```

(c)
```
struct edge {
  struct {
    bool splitted : 1;
    unsigned id : 31;
  } bits;
  pair<vertex *, vertex *> v_pair;
  pair<face *, face *> f_pair;
};
```

(d)
```
struct simplex {
  struct {
    unsigned fu_ind0 : 2;
    unsigned fu_ind1 : 2;
    bool orientation : 1;
    bool current : 1;
    unsigned id : 26;
  } bits;
  faceuse * fu_vector[4];
  simplex * parent;
  simplex * child[2];
};
```

(e)
```
struct faceuse {
  simplex * s_ptr;
  faceuse * fu_mate_ptr;
  face * f_ptr;
};
```

(f)
```
struct vertex {
  struct {
    bool current : 1;
    bool boundary : 1;
    unsigned id : 30;
  } bits;
  edge * e_ptr;
};
```

Figure 2: Modeling a BMT.

References

[1] J. Alexander. The combinatorial theory of complexes. *Ann. Math.*, 31:294–322, 1930.

[2] P. Cignoni, L. De Floriani, P. Magillo, E. Puppo, and R. Scopigno. TAn2 - visualization of large irregular volume datasets. Technical Report DISI-TR-00-07, University of Genova (Italy), 2000.

[3] T. Dey, H. Edelsbrunner, and S. Guha. Computational topology. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry (Contemporary mathematics 223)*, pages 109–143. American Mathematical Society, 1999.

[4] D. P. Dobkin and M. J. Laszlo. Primitives for the manipulation of threedimensional subdivisions. *Algorithmica*, 4:3–32, 1989.

[5] Jihad El-Sana and Yi-Jen Chiang. External memory view-dependent simplification. *Computer Graphics Forum*, 19(3), August 2000. ISSN 1067-7055.

[6] Jihad El-Sana and Amitabh Varshney. Generalized view-dependent simplification. *Computer Graphics Forum*, 18(3):83–94, September 1999. ISSN 1067-7055.

[7] Andreas Fabri, Geert-Jan Giezeman, Lutz Kettner, Stefan Schirra, and Sven Schonherr. On the design of CGAL a computational geometry algorithms library. *SP&E*, 30(11):1167–1202, 2000.

[8] L. De Floriani, E. Puppo, and P. Magillo. A formal approach to multiresolution modeling. In W. Straßer, R. Klein, and R. Rau, editors, *Theory and Practice of Geometric Modeling*. SpringerVerlag, 1996.

[9] W. B. R. Lickorish. Simplicial moves on complexes and manifolds. In *Proceedings of the Kirbyfest*, volume 2, pages 299–320, 1999.

[10] J. Peter May. *Simplicial Objects in Algebraic Topology*, volume 11. D. Van Nostrand Company, Inc., Princeton, 1967.

[11] P. Shirley and A. A. Tuchman. Polygonal approximation to direct scalar volume rendering. *Computer Graphics*, 24(5):63–70, 1990.

[12] Jeremy G. Siek, Lie-Quan Lee, and Andrew Lumsdaine. *Boost Graph Library, The: User Guide and Reference Manual*. Addison-Wesley, 2002.

[13] Jeremy G. Siek and Andrew Lumsdaine. The matrix template library: A generic programming approach to high performance numerical linear algebra. In *ISCOPE*, pages 59–70, 1998.

[14] A. A. Stepanov and M. Lee. The Standard Template Library. Technical Report X3J16/94-0095, WG21/N0482, ISO Programming Language C++ Project, 1994.

[15] B. Stroustrup. *The C++ Programming Language: Third Edition*. Addison-Wesley, 1997.

[16] Luiz Velho and Jonas Gomes. Variable resolution 4-k meshes: Concepts and applications. *Computer Graphics forum*, 19:195–212, 2000.

[17] Kevin Weiler. Edge-Based Data Structures for Solid Modeling in Curved-Surface Environments. *IEEE Computer Graphics and Applications*, 5(1):21–40, 1985.

[18] P. Williams. Visibility ordering meshed polyhedra. *ACM Transactions on Graphics*, 11(2), 1992.