Dynamic Adaptive Meshes

LUIZ VELHO

IMPA – Instituto de Matematica Pura e Aplicada
lvelho@impa.br

Abstract. In this paper we present *dynamic adaptive meshes*, a scheme for maintaining a conforming triangulation of time-varying surfaces. The user supplies an initial mesh, a surface sampling procedure and a set of adaptation criteria. The mesh is automatically modified in order to conform to user defined characteristics, while the surface changes over time. Our scheme is very simple to implement. It employs a half-edge data structure without any extra storage requirements. The mesh has an underlying semi-regular multiresolution structure. Furthermore, the specification of desired mesh characteristics can be based on very general adaptation rules.

1 Introduction

Polygonal meshes are arguably the most common representation for surfaces in geometric modeling and computer graphics.

In many applications, the surface is static and does not change after it has been created. Nonetheless, there are several types of situations where the surface is dynamic and changes its shape over time. This encompasses a wide range of applications, from the animation of deformable bodies to progressive multiscale for transmission and visualization.

There are other situations where the surface shape itself is fixed, but the surface discretization has to change for computational reasons. This occurs, for example, in the case of finite element simulations (FEM) with rigid bodies.

In the two situations described above, the application ideally should be concerned with the solution of the problem at hand, using a mesh representation only as a means to perform computations.

In practice, however, it is very common that a significant part of the effort in developing such applications goes into the task of maintaining the mesh representation. Worse yet, sometimes it is difficult to separate parts of the implementation related to the main problem domain from the mesh infrastructure, making the code less portable and error prone.

In order to overcome these drawbacks, it would be desirable to have a mesh library that could encapsulate all the functionality for supporting a dynamic mesh representation. The implementation should be robust, computationally efficient and economical in terms of space. Moreover, the application program interface (API), should be simple and provide the right level of abstraction.

In this paper we present a simple scheme for creating and maintaining a mesh representation of time-varying surfaces that addresses all the above mentioned requirements. As an additional benefit, our mesh representation has an underlying semi-regular multiresolution structure which can be exploited in various ways.

More specifically, the relevant contributions of this work are:

- The introduction of an effective mechanism for refinement and simplification of semi-regular meshes that maintains a restricted multiresolution structure. This mechanism is based on the concept of a restricted binary multi-triangulation.

- A method for maintaining a conformal mesh structure that dynamically changes its resolution based on user defined criteria. This makes the associated adaptation capabilities very general and powerful.

- The design of a minimal application program interface for mesh creation and adaptation. This interface complements the traditional topological query operators and consists of only three functions.

- The development of a simple mesh library based on the half-edge, a standard topological data structure. The implementation of this new adaptive multiresolution functionality does not require any extra storage in the representation. Also, because the half-edge is widely adopted, it should be easy to incorporate the library in many applications.

The rest of the paper is organized as follows. Section 2 reviews some basic concepts of multiresolution and Stellar theory that are used in our work. Section 3 presents binary multi-triangulations, the underlying multiresolution structure of our mesh representation. Section 4 introduces our dynamic adaptive meshes and its basic operators. Section 5 discusses the data structures used in the implementation. Finally, Section 6 gives an specification of the API of our mesh library and shows a few examples of its use.

2 Multiresolution and Stellar Theory

In this section we review multiresolution and its relationship with stellar subdivision theory.

We are concerned with a mesh representation of manifold surfaces. Furthermore, we will focus our attention on triangulated manifolds. This is not a serious restriction since every smooth manifold is triangulable [9]. In this representation, a combinatorial manifold is described by a simplicial complex, e.g. a triangle mesh.

A mesh is a cell complex $K = (V, E, F)$, formed by sets of simplices of dimension 0, 1 and 2. That is, vertices $v_i \in V$, edges $(v_i, v_j) \in E$ and triangles $(v_i, v_j, v_k) \in F$, respectively.

In a simplicial complex the cells have to meet in a "nice" way, such that: every n-dimensional face of a simplex $\sigma \in K$, is also in K; and the intersection $\sigma_1 \cap \sigma_2$ of any two simplices in K is a face of both σ_1 and σ_2. Also, the manifold representation implies that the star of every vertex $v \in V$ is a combinatorial disk.

A multiresolution mesh is a monotonically increasing sequence of simplicial complexes $H = (M_0, M_1, \ldots, M_k)$, such that $|M_i| \leq |M_j|$ for $i < j$, were $|M|$ denotes the number of triangles of M. Furthermore, the meshes $M_i \in H$ are assumed to be equivalent triangulations of a manifold surface S. Here, equivalent means piecewise linearly homeomorphic.

Any two subsequent meshes M_i, M_{i+1} in the sequence should related by a transformation that takes one into the other. In this way, the multiresolution structure can be constructed from an initial coarse mesh by successively applying refinement transformations. If the refinement transformation has an inverse, it is also possible to build the multiresolution structure starting from a dense mesh and repeatedly making simplification transformations.

Usually, it is desirable that these transformations consist of local modifications which affect only part of the mesh. In this case, the dependency relations between any two meshes in H will be local. Based on this fact, it is possible to derive from H a dependency graph that allows the extraction of new meshes not present in the original sequence.

From the above discussion on multiresolution structures we conclude that we need local operators to transform simplicial complexes. The theory of stellar subdivision provides such operators.

Stellar theory studies equivalences between simplicial complexes [7]. The main result of this theory is stated below:

Theorem 1. *([10], [11]) Two connected combinatorial n-dimensional manifolds are piecewise linearly homeomorphic if and only if they are related by a sequence of elementary stellar exchanges.*

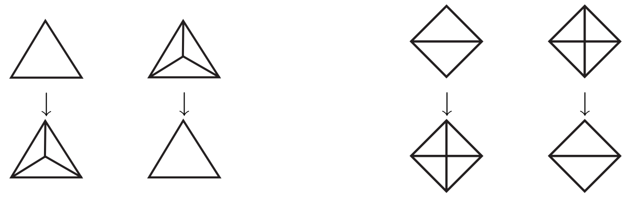
The above theorem says that stellar exchanges are the operators to transform between equivalent combinatorial manifolds.

Stellar exchanges are atomic operations that make local changes to the neighborhood of an r-simplex in a n-dimensional complex K, $r \leq n$, while maintaining the combinatorial integrity of K.

Definition 2. *Let K be a an n-dimensional simplicial complex, take a non-empty r-simplex $A \in K$, such that $\mathrm{link}(A, K) = \partial B \star L$ for some non-empty simplex $B \notin K$, and some complex L.*

The operation that changes K into K' by removing $A \star \partial B \star L$ from K and inserting $\partial A \star B \star L$ is called a stellar exchange *and is written $K \overset{\kappa(A,B)}{\longmapsto} K'$.*

The stellar exchanges operations unify the notions of bistellar moves and stellar subdivision. When K is a 2-dimensional complex we have the following operators: *face split* and its inverse *face weld*; *edge split* and its inverse *edge weld*; and *edge flip*; Note that the edge flip is its own inverse. These operations are illustrated in Figures 1 and 2.



(a) face split/weld (b) edge split/weld

Figure 1: Stellar subdivision operators.

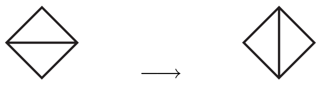


Figure 2: Bistellar move on edges – flip operator.

Stellar subdivision is an important part of the stellar theory and is relevant in the context of multiresolution because it provides the basic operators to refine and simplify a simplicial complex. Here we mention only one result of stellar subdivision theory [1]

Proposition 3. *Any stellar operation can be decomposed into a finite sequence of elementary stellar operations on edges.*

The result above allows us to restate theorem 1 only in terms of stellar operations on edges. Therefore, these operators can be used as building blocks for creating multiresolution structures.

3 Binary Multi-triangulations

As we have seen in the previous section, there are good reasons to restrict the stellar operations to moves on edges. Whenever a stellar subdivision happens in an edge ε, the triangles incident in ε are splitted into two. Accordingly, a sequence of stellar subdivisions induces a binary tree structure in the cells. Binary trees are simple and often lead to efficient algorithms.

In order to define the concept of binary multi-triangulation (BMT), we need some auxiliary definitions. Here, we follow closely the definitions in [4]. A *partially ordered set* (poset) $(C,<)$ is a set C with an 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 1. *A* binary multi-triangulation *is a poset* $(\mathscr{T}, <)$*, where* $\mathscr{T} = \{M_\lambda\}$ *is a finite set of simplicial complexes (named* triangulations*) and the order* $<$ *satisfies:*

1. *$M \prec M'$ if, and only if,* $M \overset{\kappa(\varepsilon,v)}{\longmapsto} M'$*, for some edge* $\varepsilon \in M$*, where* $\overset{\kappa(\varepsilon,v)}{\longmapsto}$ *is a non-increasing stellar exchange.*

2. *There is a maximum and a minimum complexes in* $\mathscr{T}$*, called* base triangulation *and* full triangulation*, respectively* [1]*;*

Property 2 says that a BMT is a *lattice*. Other fact which follows from the definition is that every two triangulations in $\mathscr{T}$ are stellar equivalent. 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, and nodes are sub-meshes. From an algorithmic perspective, the key idea is to use the above mentioned interleaved binary tree structures in the simplices to encode the DAG.

Any cut in the DAG that separates the source from the drain represents a valid mesh, such a cut is called *front*. Figure 3 illustrates this concept. Note that this mechanism allows the generation of a large number of meshes, distinct from the ones defined in the multiresolution sequence. It is the flexibility in choosing how to make those cuts that warrants the expression power of a BMT.

The BMT is, in fact, a particular case of a more general variable resolution structure, called Multitriangulation [5].

[1] One can replace $M' \prec M$ by $M \prec M'$ in property 1. In this case, we must interchange base triangulation and full triangulation. This is a transformation from an *increasing* to a *decreasing* BMT. [12] demonstrates that increasing and decreasing multi-triangulations are equivalent.

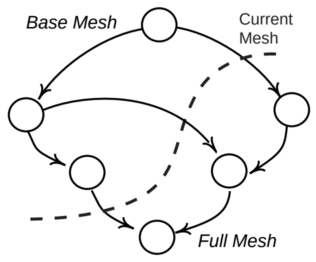


Figure 3: BMT viewed as a DAG

Binary Multi-Triangulations have been introduced in [13]. We define now a more specialized version of the BMT.

Definition 2. *A* regular binary multi-triangulation *is a binary multi-triangulation that satisfies the following conditions:*

1. *The base triangulation is a* tri-quad mesh*.*

2. *Refinement operations are only applied to interior edges of* basic blocks*.*

The requirements of a regular binary multi-triangulation (RBMT), are based on the concepts of tri-quad mesh and basic-block. A *tri-quad mesh* is a triangulated-quadrangulation, i.e., a mesh that is the union of basic quadrilateral blocks. A *basic block* is a pair of triangles with a common edge, called *internal edge* of the block. The other edges are called *external edges*. Figure 4 shows a tri-quad mesh, where the interior edges of basic blocks are rendered as dashed lines.

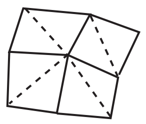


Figure 4: Example of a tri-quad mesh.

Note that, when the interior edge of a basic block is subdivided by a split operation, it is necessary to form new basic blocks in order to regularize the mesh. This is done by refining adjacent blocks. In that way, the external edges of previous blocks become internal edges of new blocks. This process is called *interleaved refinement*. (See Figure 5.)

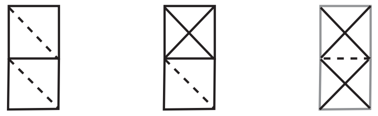


Figure 5: Interleaved refinement.

It is the interleaved refinement process that gives an underlying regular structure to the meshes extracted from a RBMT. In addition, this process guarantees that any two adjacent triangles in a mesh will differ at most by one level of refinement. This is what is called a *restricted structure*.

Thus, the RBMT can be viewed as a forest of interleaved quad-trees formed by basic blocks. The concept of regular binary multi-triangulations was introduced by [14], in connection with the $\sqrt{2}$ subdivision scheme.

The RBMT has very special properties. It is optimal in relation to the main criteria used to measure the effectiveness of a variable resolution structure (see [12]). More precisely, I achieves the highest expressive power, it has logarithm depth and also linear growth rate.

4 Adaptive Meshes

Binary Multi-triangulations possess a very rich adaptation structure. The key element for this property is the ability to make local changes in the current mesh by moving the cut around a node in the DAG.

The stellar operators split and weld implement just "local" transitions in the DAG, that is, if $\mathscr{T}$ and $\mathscr{T}'$ are the triangulations before and after split be called, respectively, then $\mathscr{T}' \prec \mathscr{T}$.

However, in order to keep the current mesh consistent it is necessary to propagate the dependencies between sub-meshes that are encoded in the DAG. This propagation mechanism is based on the following observation.

Proposition 4. *In a binary multi-triangulation, every* 2-*simplex, $\sigma \notin \overline{M} \cup \underline{M}$, not in either the base mesh $\overline{M}$ or the full mesh $\underline{M}$, has a* refining element *and an* unrefining element.

The refining element will be either a split edge e_s *or a flip edge* e_f*, while the unrefining element will be either a* weld vertex v_w*, or a flip edge* e_f.

This means that, before applying a stellar subdivision operation on some element of the mesh (e.g., an edge split or an edge weld), all triangles that are incident on the element (e.g., the edge to be refined or the vertex to be simplified) must have that element as the subdivision element (e.g., refining or unrefining).

Note also that a restricted binary multi-triangulation uses only the edge split and edge weld operators.

Proposition 4 leads automatically to a restricted hierarchical structure. Moreover, it can be implemented using a simple recursive algorithm. It is also remarkable that this algorithm is basically the same for both refinement and simplification.

Below, we present the pseudo C++ code of the BMT restricted refinement and simplification algorithms.

Program 1 Restricted edge refinement.

```
Vertex* Mesh::refine(Edge* e)
{
  for (Iterator f = incident_faces(e))
    if (f->split_edge() != e)
      refine(f->split_edge());
  return split(e);
}
```

Figure 6 gives an example of dependency propagation in restricted edge refinement.

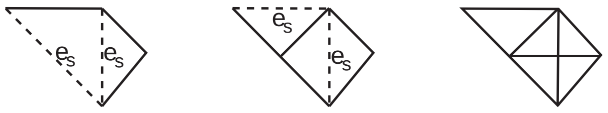


Figure 6: Dependency propagation in edge refinement.

Program 2 Restricted vertex simplification.

```
Edge* Mesh::simplify(Vertex* w)
{
  do {
    Vertex* v = max_level_neighbor(w);
    if (v->level() > w->level())
      simplify(v);
  } while (w->degree() != w->weld_degree());
  return weld(w);
}
```

In Program 2, the function max_level_neighbor(w), returns the vertex in the link of w with highest subdivision level. Note that the order of welding incident simplices to the welded vertex is relevant: they must be welded from the highest to the lowest level.

The method weld_degree returns the degree that a vertex should have for a weld be applied. This value is operator dependent – 4 for a weld of an internal edge and 3 for a weld of a boundary edge.

Figure 7 gives an example of dependency propagation in restricted vertex simplification.

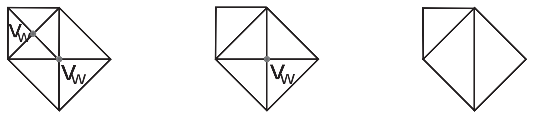


Figure 7: Dependency propagation in vertex simplification.

The two above algorithms make non-local transitions in the DAG, propagating dependencies to maintain the mesh consistent. In order to perform global adaptation on a mesh, it is necessary to apply these two algorithms on the whole mesh and refine or simplify it where desired.

The code for global adaptive refinement and simplification is shown in Programs 3 and 4, respectively.

Program 3 Adaptive mesh refinement.

```
void Mesh::adapt_refine(Function* need_ref)
{
  bool changed;
  do {
    changed = false;
    for (e = edges_begin(); e != edges_end(); e++) {
      if (e->is_split() && need_ref(e)) {
        refine(e);
        changed = true;
      }
    }
  } while(changed);
}
```

Program 4 Adaptive mesh simplification.

```
void Mesh::adapt_simplify(Function* need_simpl)
{
  bool changed;
  do {
    changed = false;
    for (v = vertices_begin(); v != vertices_end(); v++) {
      if (v->is_weld() && need_simpl(v)) {
        simplify(v);
        changed = true;
      }
    }
  } while (changed);
}
```

We remark that, as in the restricted simplification methods, the adaption algorithm is essentially the same for refinement and simplification. The algorithm keeps scanning through the appropriate transition elements for the operation (i.e., edges for refinement and vertices for simplification). For each valid transition element (split or weld), it asks if the transition should be made and performs the operation when this condition is true.

There are a couple of observations regarding the above algorithms. First, observe that stellar operations are independent of each other because they change only the 1-neighborhood of the element. Each operation automatically maintains the restricted structure.

The user supplies the adaptation criteria through a function that is called with the transition element as its argument. This function can be very general and can be based either on local information from the mesh – queried through the element, or any other global information – can depend on other factors besides the mesh itself, such a camera position in a view dependent adaptation. Also, the function is responsible to make sure that the criterium will eventually be satisfied so that the process terminates.

In order to have a complete adaptation, part of the mesh might need to be refined, while other parts may have to be simplified. Consequently, the two functions should be called every time the conditions change. as illustrated in the code below.

```
void Mesh::adapt(Function* ref_test, Function* simpl_test)
{
  adapt_refine(ref_test);
  adapt_simplify(simpl_test);
}
```

In this full adaptation case, it is mandatory that the tests for refinement and simplification are based on the same adaptation criteria and complement each other. This guarantees that the final result will be correct.

A simple example is as follows. Suppose that the criteria is to keep the mesh resolution at level J. Then, the test in need˙ref(e) should be $(e->level() < J)$, while the test in need˙simpl(w) should be $(w->level() > J)$.

5 Data Structures

The algorithms described in the previous section are applicable to general BMT's, and also to regular BMT's.

However, in the case of general BMT's, it is necessary to store explicitly the dependency graph. This DAG essentially encodes the transition elements for refinement and simplification, as well as, the geometric information associated with changes of mesh resolution. A hierarchical data structure suitable for this purposed has been introduced in [13].

In this section, we exploit the properties of the regular BMT, in order to provide all the functionality discussed in Section 4, without the need for a hierarchical data structure.

To motivate the applicability of our scheme, we observe that in many practical situations shape information is known independently of the mesh. This is particularly true when the surface shape is not static and changes dynamically over time. Some examples are variational modeling, multiresolution editing, surface remeshing and visualization of parametric or implicit surfaces.

Typically in such applications, the following requirements are satisfied:

- There is a base domain where the surface geometry is defined;
- It is possible to compute sample points on the surface.

In this case, it would be beneficial to have a mesh library that could take care of maintaining an adaptive variable resolution mesh, but storing only the current mesh and providing a simple interface.

The key to achieve this goal is to impose a regular binary multi-triangulation structure to the mesh.

The user must provide the topology and geometry of an initial base polyhedron – that could be specified in any standard format, such as OFF (Object File Format). From this description, the library constructs the dynamic mesh structure, initialized as a tri-quad mesh. An algorithm to create tri-quad meshes is given in [14]. We include it here for completeness.

Program 5 Construction of a tri-quad mesh.

```
void Mesh::make_triquad(void)
{
  priority_queue<node,vector<node>,less<node>> q;
  for (e = edges_begin(); e != edges_end(); e++) {
    e->set_mark(false);
    q.push(node(e));
  }
  while ( !q.empty() ) {
    node n = q.top(); q.pop();
    if ( !n.edge->is_marked() ) {
      mark_link(n.edge);
      split(n.edge);
    }
  }
  for (f = faces_begin(); f != faces_end(); f++)
    if (f->level() == 0)
      split(f);
}
```

The tri-quad mesh generation algorithm shown in Program 5 works as follows: the mesh is decomposed into clusters of triangle-pairs and isolated single triangles. The boundaries of these clusters will form the internal edges of basic blocks in the tri-quad mesh. This is done by subdividing triangle pairs with the edge split operator and single triangles with the face split operator. Note that, in this algorithm, triangle pairs are created based on longest edge clustering.

Obviously, when the input mesh already possesses a quadrilateral structure, it is not necessary to apply the tri-quad mesh algorithm. The library provides a way to bypass this initial phase and construct the base mesh directly from the quadrangulation.

The library uses an standard representation based on the half-edge data structure. The specification of our mesh representation is shown in Figure 8.

In this representation, a mesh is a collection of sets of vertices, edges and faces. The geometric data for refining the mesh is obtained through a user-supplied sampling function. A triangular face is a loop of three oriented edges. An edge stores two half edges, one for each orientation. The half-edge contains pointers to its origin vertex, next half-edge in the loop, face corresponding to that loop, and its parent edge. The vertex stores its subdivision level, and a pointer to one of the incident edges in its star.

```
struct Mesh {
  Container<Face*>    faces;
  Container<Edge*>    edges;
  Container<Vertex*>  vertices;
  Function*           sample;
}

struct Face {
  Half_Edge*  he_ref;
}

struct Edge {
  Half_Edge   he[2];
}

struct Half_Edge {
  Vertex*     org_ref;
  Half_Edge*  nxt_ref;
  Face*       f_ref;
  Edge*       e_ref;
}

struct Vertex {
  int         level;
  Half_Edge*  star;
}
```

Figure 8: Half-edge based mesh representation.

In order to refine and simplify the current mesh, it is necessary to know the split edge and weld vertex of a triangle. We want to obtain this information without having to store the data explicitly. This is possible because of the regular structure of the RBMT. Note that in an RBMT, the split edge is always opposite to the weld vertex, as shown in Figure 9. Therefore, we establish a convention that the split edge and the weld vertex are numbered as the first simplices of dimension 1 and 0 of a triangle, i.e., $e_s = e_0 \in (e_0, e_1, e_2)$ and $v_w = v_0 \in (v_0, v_1, v_2)$. This numbering is consistent with the face operator definition for abstract simplicial complexes [8]. The scheme is automatically maintained by our library's implementation of the stellar operators.

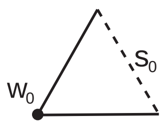


Figure 9: Numbering scheme for subdivision elements.

With the numbering scheme above we have all the information necessary to implement the split operator. However, the weld operator requires one more piece of information. It is given a weld vertex *w* and must transform the star of *w* into two triangles. Thus, it needs to know which vertices from the link of *w* should be connected to create the new internal edge of the block. We encode this information, using the convention that the origin vertex of the welded edge is the same as the origin of the half-edge indicating the star of *w*. This scheme is shown in Figure 10.

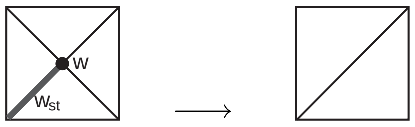


Figure 10: Scheme for encoding the weld of an edge.

The mesh representation also allows all the standard topological queries for navigating a combinatorial manifold structure. Most operators are easily inferred by inspection.

In particular, the data structure encodes the relevant information as follows:

- Each element has sufficient information to recover its incident elements and its star;
- The split edge of a triangular face is given by face.edges(0).
- The weld vertex of a triangular face is given by face.vertices(0). Note that this is a consequence of the previous item.
- The star of a vertex is indicated by a pointer to one of its incident half-edges. This half edge also indicates the origin of the welded edge.
- An edge is on the boundary if, and only if, e.star.mate.face() == NULL. There is an analogous condition for a boundary vertex.

Actually, the data structure exhibited in Figure 8 is not exactly the same we use in our implementation. The main difference is that the main class is parametrized by an *attribute class*, which enables that new attributes be added to any element. One can, for instance, add the surface normal or a scalar value to each vertex. This is done in compile time and causes no storage overhead. A similar technique is described in [2].

In summary, our mesh representation uses a standard topological data structure, and is able to encode implicitly all information necessary for maintaining a dynamic variable-resolution mesh with the underlying structure of a RBMT.

6 API and Examples

In this section we describe the adaptive mesh library's API (Application Program Interface), and give examples of using the library in simple applications.

The library API, related to mesh construction and adaptation is composed of only three functions:

- Mesh(Function* sample) – this function is the mesh constructor.
- Mesh::read˙off(string filename) – this function reads a polyhedron describing the surface domain, and creates a base mesh with a triangulated quadrangulation structure. The function automatically detects whether the tri-quad algorithm should be applied or not, i.e., quadrilateral faces of the polyhedron will correspond to basic blocks in the base mesh.
- Mesh::adapt(Function* ref˙tst, Function* simp˙tst) – this function simply calls adapt˙refine and adapt˙simplify with the proper parameters. These two adaptation functions are also exposed by the library API, in case the application needs only refinement or simplification.

As seen above, the API requires that the application supplies three functions: a *sampling function* to evaluate the geometry of the surface at vertex of the mesh; a *refinement test* and a *simplification test* to determine if the mesh needs further subdivision or coarsening, respectively.

Program 6 shows the pseudo C++ code for the basic structure of a typical application using the library. First a new mesh is created and the base mesh is initialized. Then, the application enters in the processing–adaptation loop, where a user function that updates the mesh is called, followed by a call to mesh adapt.

Program 6 Application main loop.

```
Mesh m = mesh(sampl_func);
m.read_off(filename);
while ( do_processing() ) {
  update_mesh(m);
  m.adapt(ref_test, simpl_test);
}
```

In addition to the dynamic adaptation API, the library also provides the standard operators for querying and navigating topological data structures.

Note that the data structure does not include any geometric data. This is supplied by the application through an attribute class. The library deals only with topological issues (except during the construction of the base mesh when the geometry is provided in the OFF file). Therefore, the application has full control of geometry aspects through the

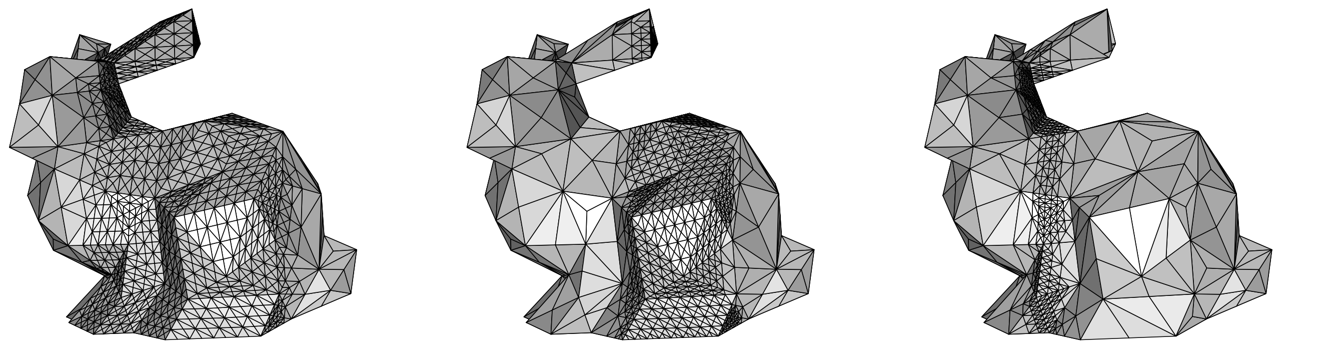


Figure 11:

vertex attribute class and the sampling function. This mechanism makes the library more general, with the flexibility to accommodate different types of surface description, such as parametric, implicit, etc. It is worthwhile noting that the regular structure of the RBMT implies that every vertex V can be labeled with a unique index (b,i,j,k), where b indicates the base triangle from which that vertex was generated. Other labeling schemes are possible when the surface has a particular form of parametrization.

Figure 11 shows snapshots of a simple application using the library. In this program, a coarse model of the Stanford Bunny is tessellated with different resolutions that are controlled by the user in real-time using the mouse. The program was implemented as plug-in for Geomview and has only 97 lines of code, in addition of the mesh library.

References

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

[2] Mario Botsch, Stephan Steinberg, Stephan Bischoff, and Leif Kobbelt. Openmesh - a generic and efficient polygon mesh data structure. In *OpenSG PLUS Symposium*, 2002.

[3] 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.

[4] 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.

[5] Leila De Floriani, Paola Magillo, and Enrico Puppo. Efficient implementation of multi-triangulations. In David Ebert, Hans Hagen, and Holly Rushmeier, editors, *IEEE Visualization '98*, pages 43–50, 1998.

[6] M. Garland. Multiresolution modeling: Survey & future opportunities. In *Eurographics '99, State of the Art Report (STAR)*, 1999.

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

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

[9] J. Munkres. *Elementary Differential Topology*. Princeton University Press, 1966.

[10] M. H. A. Newman. On the foundations of combinatorial analysis situs. *Proc. Royal Acad.*, 29:610–641, 1926.

[11] U. Pachner. Pl homeomorphic manifolds are equivalent by elementary shellings. *Europ. J. Combinatorics*, 12:129–145, 1991.

[12] E. Puppo. Variable resolution triangulations. *Computational Geometry Theory and Applications*, 11(34):219–238, 1998.

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

[14] Luiz Velho and Denis Zorin. 4-8 subdivision. *Computer-Aided Geometric Design*, 18(5):397–427, 2001. Special Issue on Subdivision Techniques.