Adjacency and incidence framework: a data structure for efficient and fast management of multiresolution meshes

Conference Paper · January 2003

DOI: 10.1145/604471.604503 · Source: DBLP

2 authors:

Frutuoso G. M. Silva
Universidade da Beira Interior
50 PUBLICATIONS **265** CITATIONS
SEE PROFILE

Abel JP Gomes
Universidade da Beira Interior
120 PUBLICATIONS **750** CITATIONS
SEE PROFILE

Adjacency and Incidence Framework - A data structure for efficient and fast management of multiresolution meshes

Frutuoso G. M. Silva[*]   Abel J. P. Gomes[†]

IT - Networks and Multimedia Group
Universidade da Beira Interior

Abstract

This paper introduces a concise and responsiveness data structure, called AIF (Adjacency and Incidence Framework), for multiresolution meshes, as well as a new simplification algorithm based on the planarity of neighboring faces. It is an optimal data structure for polygonal meshes, manifold and non-manifold, which means that a minimal number of direct and indirect accesses are required to retrieve adjacency and incidence information from it. These querying tools are necessary for dynamic multiresolution meshing algorithms (e.g. refinement and simplification operations). AIF is an orientable, but not oriented, data structure, i.e. an orientation can be topologically induced as needed in many computer graphics and geometric modelling applications. On the other hand, the simplification algorithm proposed in this paper is ”memoryless” in the sense that only the current approximation counts to compute the next one; no information about the original shape or previous approximations is considered.

CR Categories: I.3.5 [Computer Graphics]: Boundary representations—Curve, surface, solid and object representation; I.3.6 [Methodology and Techniques]: Graphic data structure and data types

Keywords: boundary representation, polygonal meshes, multiresolution algorithms, mesh simplification

1 Introduction

The complexity of 3D object models increases everyday in part due to more and more demanding interactive 3D design tools, data acquisition technologies, geometric processing, and graphics workstation capabilities. In the web scenario, the need to manipulating these models through coding, transmission, visualization, and storage operations compel us to develop better mesh simplification and refinement algorithms in order to cope with huge amounts of geometric data and responsiveness.

This problem can be partially resolved by using multiresolution meshes. Multiresolution allows us to render a mesh at different levels of resolution, depending on its projective distance to a virtual viewer. For that, we use simplification and refinement operations on polygonal meshes, by coalescing and subdividing vertices, edges and faces, respectively.

However, simplification and refinement operations require fast access and retrieval of local adjacency and incidence information from mesh data structures. For example, it is often necessary to identify the neighboring vertices around a vertex, or which faces are incident on an edge. Unfortunately, these operations are not as fast and efficient as necessary, in particular for meshes with a huge amount of cells (say, vertices, edges, and faces).

This paper introduces a new data structure, called AIF (Adjacency and Incidence Framework), and its associated query operator in order to satisfy two major important requirements, namely: conciseness and responsiveness. In a more enlarged setting, this paper addresses the following issues:

- *Generality*. AIF does not only support triangular meshes. It may accommodate general polygonal meshes, regardless of whether they are manifold or not.

- *Conciseness*. Unlike other b-rep (boundary representation) data structures, AIF is not oriented, i.e. it does not contain oriented cells (e.g. half-edges or co-edges). Consequently, it is then more concise than the traditional b-rep meshing data structures.

- *Orientability*. Despite the absence of oriented cells, AIF is an orientable data structure as needed in, for example, rendering applications. Each face normal is computed by consistently inducing a topological orientation along its frontier. By a consistent topological orientation we mean that all faces are counter-clockwise oriented.

- *Scalability*. AIF admits multiple levels of detail or scale, what is accomplished by meshing refinement and simplification algorithms. This enables a device-adapted visualization, depending on its graphic resolution.

- $Sub-meshing$. AIF should provide a sub-meshing mechanism to deal with mesh regions as required in interactive modelling and animation. But, this topic will be addressed in a later paper.

- *Responsiveness*. AIF was designed to ensure fast queries through a single indexed query operator, called mask operator. It basically returns adjacency and incidence data.

This paper is organized as follows. Related work appears in Section 2. Section 3 describes the AIF representation. Section 4 presents the implementation of the AIF data structure. Section 5 describes its adjacency and incidence operator. Section 6 deals with a new simplification algorithm for multiresolution meshes. Section 7 presents some relevant experimental results. Lastly, Section 8 draws some conclusions and future work.

[*]e-mail:fsilva@di.ubi.pt
[†]e-mail:agomes@di.ubi.pt

2 Related work

Much research on polygonal meshes has been done in the last few years, particularly in multiresolution analysis [Eck et al. 1995; Lounsbery et al. 1997; Garland and Heckbert 1997; Hubeli and Gross 2000], simplification and refinement of polygonal models [Garland 1999; Hoppe et al. 1993], view-independent LOD and selective refinement [Hoppe 1998; Kim and Lee 2001], mesh morphing [Lee et al. 1999], interactive mesh edition [Zorin et al. 1997] and even in geometric compression and transmission over the web [Taubin and Rossignac 1998; Touma and Gotsman 1998].

Many data structures are not appropriate for the design and implementation of fast and efficient traversal algorithms over very large meshes. For example, finding a cell in the cell-tuple data structure [Brisson 1993] implies processing all cell-tuples, a time-consuming operation in particular for large meshes.

Related data structures represent a triangular mesh by a set of faces, each face being composed by a tuple of vertices [Hoppe 1998], as it is the case of VRML [Carey et al. 1997] (Virtual Reality Modelling Language) file-format. This makes it difficult to find the faces incident at a given vertex because we risk traversing all the faces in the data structure. The performance of this lookup algorithm gets even worse for large meshes.

A way to speed up lookup algorithms is to use oriented boundary representation data structures (winged-edge [Baumgart 1972], half-edge [Mantyla 1988] or radial-edge [Weiler 1998]), but they need additional storage space for oriented cells. For example, in the radial-edge data structure, each face has two associated loops, one for each face side; hence, three faces incident along an edge requires six oriented edges.

Other data structures were designed only for triangular meshes as, for example, Directed Edges [Campagna et al. 1998] or Tri-Edges [Loop 2000] or, more generally, n-dimensional simplicial complexes as the PSC [Popovic and Hoppe 1997] (Progressive Simplicial Complexes) data structure. The first two are oriented data structures, while the PSC data structure is non-oriented. This means that PSC can be used to represent orientable and non-orientable objects because oriented simplexes are absent from PSC. Thus, unlike oriented b-rep data structures, there is no simplex redundancy in PSC data structure. However, the inexistence of explicit oriented simplexes in PSC poses some difficulties in rendering. In fact, unlike progressive meshes proposed by [Hoppe 1996], the PSC data structure explicitly avoids storing surface normals at vertices. Instead, it makes usage of smoothing group fields for different materials as used by Wavefront Technologies [1998].

This paper proposes the AIF data structure for generic meshes. These meshes need not be manifold or triangular. Thus, it can accommodate simplicial complexes and, more generally, cell complexes with or without dangling cells. Topologically speaking, it is an orientable, but not oriented, data structure, i.e. it does not possess oriented cells. A geometric orientation or surface normals are computed by inducing a topological orientation on frontier cells of each face, consistently. The result is a more concise and flexible data structure, without losing control on the local adjacency and incidence information.

3 Adjacency and Incidence Framework

It is said that a cell x is adjacent to a cell y (symbolically, $x \prec y$) if x is contained in the frontier of y (or, equivalently, $fr(y) \cap x \neq \emptyset$) and the dimension of x is lesser than the dimension of y; equivalently, one says that y is incident on x (symbolically, $y \succ x$). For example, a vertex bounding an edge is said to be adjacent to it, but there may be many edges incident at the same vertex. The adjacency (and incidence) relation is transitive, i.e. if $v \prec e$ and $e \prec f$ then $v \prec f$.

Besides, there are only two basic adjacency relations for 2-dimensional cell complexes, namely $V \prec E$ and $E \prec F$; $E \prec V$ and $F \prec E$ are their inverse or incidence relations. These four basic relations can be compounded to form nine adjacency relations as introduced by Weiler [1985]. For example, the $V \rightarrow F$ and $E \rightarrow E$ relations introduced by Weiler can be obtained as follows:

$$V \rightarrow F = (V \prec E) \circ (E \prec F)$$
$$E \rightarrow E = (E \succ V) \circ (V \succ E)$$

According to Ni and Bloor [1994], those four basic relations form the best representation in the class C_4^9 in terms of information retrieval performance. That is, it requires a minimal number of direct and indirect accesses to the data structure to retrieve those four basic adjacency and incidence relations and the remaining five compound adjacency and incidence relations, respectively. A direct access query involves a single call to the mask operator, while an indirect access requires two or more calls to the mask operator, i.e. a composed query. (The mask operator is described in Section 5.)

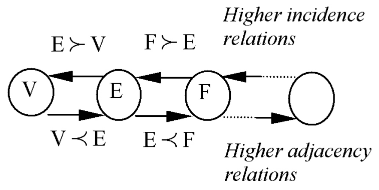

Figure 1: AIF diagram.

The AIF (Adjacency and Incidence Framework) data structure just accommodates the optimal C_4^9 representation for 2-dimensional meshes, but it can be easily generalised to $C_{2n}^{(n+1)^2}$ for n-dimensional objects (Figure 1). It keeps the essential adjacency and incidence data that allows us to derive supplementary data to traverse any mesh, triangular or otherwise.

A 2-dimensional mesh is defined by means of a triple $M = \{V, E, F\}$, where V is a finite set of vertices, E is a finite set of edges, and F is a finite set of simply connected faces.

A vertex $v \in V$ is defined by $v = \{e_1, e_2, e_3, ..., e_k\}$, where e_i, i=1,...,k, is an incident edge at v. This is the way the $E \succ V$ relation is embedded in the AIF.

An edge $e \in E$ is defined by $e = \{\{v_1, v_2\}, \{f_1, f_2, ..., f_k\}\}$, where $\{v_1, v_2\}$ is the set of its bounding (or adjacent) vertices, and $\{f_1, f_2, ..., f_k\}$ are its incident faces. So, we have $V \prec E$ and $F \succ E$ relations.

A face $f \in F$ is defined by $f = \{e_1, e_2, e_3, ..., e_k\}$, where each e_i, i=1,...,k, is an edge bounding f. This definition includes the $E \prec F$ relation.

With these four basic adjacency and incidence relations embedded in the AIF, we are able to represent manifold (Figures 2(a) and (b)) and non-manifold polygonal (Figures 2(c) and (d)) meshes. The non-manifoldness of the plane mesh in Figure 2(c) comes from the fact that some faces of its cockpit have been removed. Also, note that the AIF can accommodate general polygonal meshes, regardless of whether they are triangular (Figures 2(a) and (b)) or not (Figure 2(c)). Hence, the generality of the AIF data structure.

4 Data Structure Implementation

The AIF data structure represents a mesh (or cell complex) consisting of a set of cells (vertices, edges and faces). It basically consists of four C++ classes: `Vertex`, `Edge`, `Face`, and `Mesh`. The class `Mesh` implements a 2-dimensional mesh as a whole. The remaining classes implement cells of dimension up to 2. In general, all cell types can be implemented by a single C++ class, called `Cell`. In this case, every n-dimensional cell would be seen as composed by a set of adjacent (n-1)-cells and a set of incident (n+1)-cells. The AIF data structure is then as follows:

```
typedef vector <Vertex*> vvector;
typedef vector <Edge*> evector;
typedef vector <Face*> fvector;

class Vertex {
  evector li;        // incident edges
  Point *pt;         // geometry
  Point *nv;         // vertex normal
}

class Edge {
  Vertex *v1, *v2;  // adjacent vertices
  fvector li;        // incident faces
}

class Face {
  evector la;        // adjacent edges
}

class Mesh {
  int ID;            //mesh ID
  vvector vv;        //vector of vertices
  evector ev;        //vector of edges
  fvector fv;        //vector of faces
}
```

Note that the AIF data structure is not topologically-oriented provided that it does not include any oriented cells. It is geometrically-oriented by vertex normal `nv` in the class `Vertex`. Each vertex normal is determined from the face normals around it.

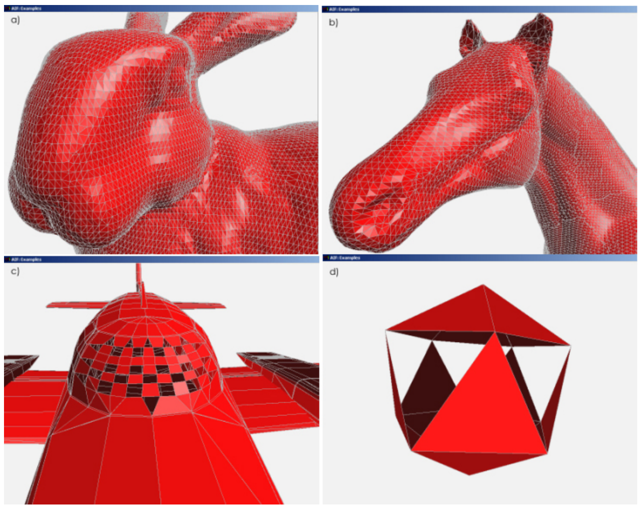

Figure 2: AIF polygonal meshes.

5 Adjacency and incidence operator

Responsiveness is a critical issue in multiresolution meshing. We need fast query and retrieval algorithms to locally identify cells adjacent to or incident at/on other cells in order to quickly refine and simplify meshes. For that, we use a single operator, called mask operator, as described below.

Before proceeding, let us first pay attention to the AIF incidence scheme. The incidence scheme of a mesh can be described in terms of a set $T = \{(v_i, e_j, f_k)\}$ of cell-tuples, where f_k is a face incident on an edge e_j (symbolically, $f_k \succ e_j$) and e_j is incident at a vertex v_i ($e_j \succ v_i$); alternatively, we say that v_i is adjacent to e_j ($v_i \prec e_j$), and e_j is adjacent to f_k ($e_j \prec f_k$). For example, the incidence scheme for the tetrahedron in Figure 3 is as follows:

(v_1, e_1, f_2)	(v_1, e_1, f_3)	(v_1, e_2, f_3)
(v_1, e_1, f_2)	(v_1, e_1, f_3)	(v_1, e_2, f_3)
(v_1, e_2, f_1)	(v_1, e_3, f_2)	(v_1, e_3, f_1)
(v_2, e_1, f_2)	(v_2, e_1, f_3)	(v_2, e_6, f_2)
(v_2, e_6, f_4)	(v_2, e_4, f_3)	(v_2, e_4, f_4)
(v_3, e_2, f_1)	(v_3, e_2, f_3)	(v_3, e_4, f_3)
(v_3, e_4, f_4)	(v_3, e_5, f_1)	(v_3, e_5, f_4)
(v_4, e_3, f_1)	(v_4, e_3, f_2)	(v_4, e_5, f_1)
(v_4, e_5, f_4)	(v_4, e_6, f_2)	(v_4, e_6, f_4)

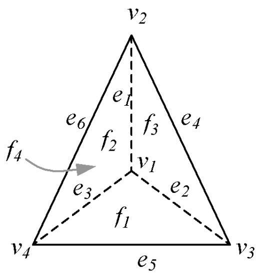

Figure 3: A triangular manifold mesh without boundary.

This incidence scheme can be considered as a data structure on its own. It is known as the cell-tuple data structure, and is due to Brisson [1993]. A nice property of the cell-tuple data structure is that only a single indexed operator it necessary to retrieve the essential adjacency and incidence information from a mesh. Such an indexed operator is the bowtie operator $\bowtie_d : T \to T$ defined by $\bowtie_d (t_{0,i}, t_{1,i}, t_{2,i}) = U$ where $U = (u_{0,j}, u_{1,j}, u_{2,j})$ is a cell-tuple subset of T such that $t_{d,i} \neq u_{d,j}$ and $t_{k,i} = u_{k,j}, \forall (u_{0,j}, u_{1,j}, u_{2,j}) \subset U$ for $d \neq k$ and $d, k = 0, 1, 2$. The indices d, k identify the 0-, 1-, 2-components of each cell-tuple, i.e. vertices, edges, and faces. The index i identifies the i-th cell-tuple in T, while the index j identifies the j-th cell-tuple in U.

In other words, applying the bowtie operator to a cell-tuple in T has as a result a subset $U \subset T$ of cell-tuples, from which one can retrieve relevant adjacency/incidence information for multiresolution meshes. Note that there is only one bowtie operator for querying and accessing this adjacency/incidence data. The bowtie operator index determines the semantics of data to be retrieved. For example, the values 0, 1, and 2 for the index i denote the intent of somehow retrieving vertices, edges, and faces, respectively. Let us see some examples for the object in Figure 3:

1. $\bowtie_0 (v_1, e_1, f_2) = \{(v_2, e_1, f_2)\}$, from which we can get the vertex v_2 bounding (adjacent to) the edge e_1.

2. $\bowtie_1 (v_1, e_1, f_2) = \{(v_1, e_3, f_2)\}$, from which we can get the edges incident at v_1 and adjacent to f_2.

3. $\bowtie_2 (v_1, e_1, f_2) = \{(v_1, e_1, f_3)\}$, from which we can get the faces incident on e_1.

To traverse the frontier of a face, say f, we alternately apply the operators $\bowtie_0$ and $\bowtie_1$ to the cell-tuple that outputs from each operator. The operator $\bowtie_0$ gives us the next vertex in the frontier of f, and the operator $\bowtie_1$ provides the next edge bounding f. This means that we use the bowtie operator as many times as the number of cells bounding f. Obviously, this slows down the navigation through a mesh because we have to traverse the cell-tuple data structure fully whenever the bowtie operator is called.

The bowtie operator is fast for small and medium meshes. But, its time performance decreases for large meshes. This is because it processes all the cell-tuples of a mesh in order to determine those having the same fixed components as the cell-tuple argument.

To prevent traversing the cell-tuple data structure fully, we have designed the AIF data structure with the same adjacency and incidence descriptive power as the cell-tuple data structure, but less space and less time-consuming. In fact, the AIF (Figure 1) consists of a set of cells (not a set of cell-tuples), with each cell defined as follows:

- A vertex is defined in terms of its incident edges;
- An edge is defined in terms of its bounding vertices and incident faces;
- A face is defined in terms of its frontier edges.

The AIF data structure also uses a single adjacency and incidence operator, called mask operator. The mask operator is defined by $\blacktriangleright\!\lhd_d : V \times E \times F \rightarrow C$, with $C = V \cup E \cup F$ being the union of the set V of vertices, the set E of edges, and the set F of faces, such that $\blacktriangleright\!\lhd_d(v_i, e_j, f_k) = \{c_1^d\}$, i.e. a set of d-dimensional cells.

The arguments of $\blacktriangleright\!\lhd_d$ are cells in set $V \times E \times F$. A NULL cell as argument of dimension $n = d$ means that all the n-cells satisfying the adjacency/incidence condition expressed by the cell arguments are to be returned; otherwise, if $n \neq d$ and the n-cell argument is also NULL, no n-cell imposes any adjacency/incidence restriction on the d-cells to be returned. In case $n = d$ and the n-cell argument is not NULL, the operator $\blacktriangleright\!\lhd_d$ returns all the n-cells as before, except the n-cell argument; if $n \neq d$ and the n-cell is not NULL, the n-cell imposes an additional adjacency/incidence restriction on the retrieved d-cells.

Let us consider again the mesh in Figure 3 to illustrate how the mask operator works in conjunction with the AIF data structure:

1. $\blacktriangleright\!\lhd_1(v_1, NULL, NULL) = \{e_1, e_2, e_3\}$ directly returns all edges incident at v_1.

2. $\blacktriangleright\!\lhd_2(v_1, NULL, NULL) = \{f_1, f_2, f_3\}$ indirectly returns all faces incident at v_1. This requires an intermediate call to $\blacktriangleright\!\lhd_1(v_1, NULL, NULL)$ to return all edges e_1, e_2, e_3 incident at v_1. Then, the operator $\blacktriangleright\!\lhd_2(NULL, e_i, NULL)$ is called for each edge e_i (i =1,2,3) in order to compute faces incident on e_i and v_1.

3. $\blacktriangleright\!\lhd_0(NULL, e_1, NULL) = \{v_1, v_2\}$ directly returns bounding vertices of e_1.

4. $\blacktriangleright\!\lhd_2(NULL, e_1, NULL) = \{f_2, f_3\}$ directly returns faces incident on e_1.

5. $\blacktriangleright\!\lhd_0(NULL, NULL, f_1) = \{v_1, v_3, v_4\}$ indirectly returns all vertices bounding f_1. This requires an intermediate call to $\blacktriangleright\!\lhd_1(NULL, NULL, f_1)$ to first determine all edges bounding f_1. Then, the operator $\blacktriangleright\!\lhd_0(NULL, e_i, NULL)$ is called for each edge e_i in order to determine vertices bounding e_i and f_1.

6. $\blacktriangleright\!\lhd_1(NULL, NULL, f_1) = \{e_2, e_3, e_5\}$ directly returns all edges bounding f_1.

The mask operator enables the fast and local management of meshes. Unlike the bowtie operator, there is no need to handle all the data structure constituents. For example, to retrieve the vertices bounding an edge, we have only to access its instance variables `v1`, `v2`. On the contrary, using the cell-tuple data structure, such a retrieval operation requires traversing all its constituent cell-tuples. Thus, the operational performance of the mask operator holds independently of the mesh size. This is particulary important for handling large meshes through refinement and simplification operations.

6 Simplification algorithm

Multiresolution modelling has received increasing attention in recent years. The most common methodologies in surface multiresolution are refinement and decimation (or simplification). A refinement algorithm iteratively adds cells to an initial coarse approximation to produce a finer approximation. A decimation algorithm is the opposite. It starts with a fine approximation and terminates with a coarser approximation by iteratively removing some cells.

There are several algorithms for simplifying a polygonal mesh (see [Garland 1999; Heckbert and Garland 1997; Cignoni et al. 1997; Cohen et al. 1996; Turk 1992; Schroeder et al. 1992] for more details). A major class of these algorithms is based on the contraction of vertex pairs [Kobbelt et al. 1998; Popovic and Hoppe 1997; Hoppe 1996; Klein et al. 1996], also known as edge collapse. Within this class, the main difference between the algorithms is the criterion chosen to collapse an edge. This criterion determines the processing time of the simplification algorithm.

Table 1 shows the total processing time of five simplification algorithms based on distinct edge collapse criteria. Hoppe algorithm [1996] is based on the minimization of an energy function. Guéziec algorithm [1995] is based on keeping a tolerance volume of a given mesh. Garland and Heckbert algorithm [1997] follows a geometric criterion that is based on the estimation of the distance to a set of planes. Lindstrom and Turk algorithm [1998] uses linear constraints primarily based on conservation of volume and area. Unlike other algorithms, Lindstrom and Turk algorithm makes decisions only based on the current approximation (no information about the original shape is used).

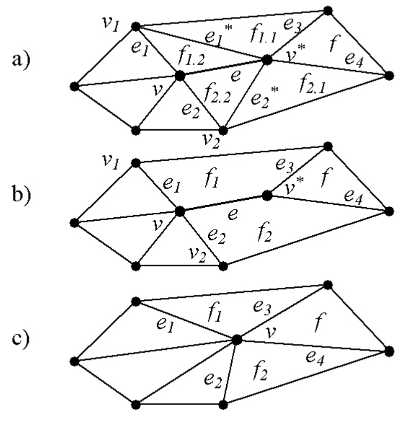

Figure 4: Mesh simplification by collapsing an edge.

The fifth entry in Table 1 concerns our Bi-Star Planarity algorithm. It is based on the planarity of face stars around the vertices bounding the edge to be collapsed. This edge collapse criterion has proven to be fast (Table 1 and Figures 5 and 6). It is a purely geometric criterion, with the collapsing edge chosen as follows:

1. Let us take an edge e;
2. If at least a vertex bounding e has been previously created by a collapse edge operation, go back to step 1 to process another edge;
3. If the faces incident on e are not approximately coplanar within a small error ε, go back to step 1 to process another edge;
4. For each vertex bounding e, if its incident faces are not coplanar within a small error $\delta > \varepsilon$, go back to step 1 to process another edge;
5. The edge e is collapsed into a new vertex (midpoint of e), and its incident faces are removed.

The edge collapse algorithm (step 5 above) can be described as follows (Figure 4):

Algorithm 1
INPUT:
(a) an edge e
Begin

1. Determine faces incident on e by calling $\blacktriangleright\!\lhd_2(NULL, e, NULL) = \{f_{1,2}, f_{2,2}\}$.
2. Select a vertex v^* bounding e.
3. Determine the edge (not e) incident at v^* and bounding $f_{1,2}$,by calling $\blacktriangleright\!\lhd_1(v^*, e, f_{1,2}) = \{e_1^*\}$.
4. Determine the edge (not e) incident at v^* and bounding $f_{2,1}$, by calling $\blacktriangleright\!\lhd_1(v^*, e, f_{2,1}) = \{e_2^*\}$.
5. Determine faces incident on e_1^* by calling $\blacktriangleright\!\lhd_2(NULL, e_1^*, NULL) = \{f_{1,1}, f_{1,2}\}$.
6. Coalesce $f_{1,1}$,$f_{1,2}$, and e_1^* into the face f_1.
7. Update adjacency and incidence relations of modified cells processed in (6).
8. Determine faces incident on e_2^* by calling $\blacktriangleright\!\lhd_2(NULL, e_2^*, NULL) = \{f_{2,1}, f_{2,2}\}$.
9. Coalesce $f_{2,1}$,$f_{2,2}$, and e_2^* into the face f_2.
10. Update adjacency and incidence relations of modified cells processed in (9).
11. Coalesce e and its bounding vertices v, v^* into v.
12. Update adjacency and incidence relations of cells changed in (11).

End

Similar to the Lindstrom and Turk algorithm, this method is ”memoryless” in the sense that it only uses the current approximation to compute the next one; no information about the original shape or previous approximations is considered. It produces very good approximations without changing the topology of the model. It also keeps the original visual shape of a model even after reducing the number of faces by as much as 90% (see Figure 9). Besides, for a given error ε, it produces a finite number of approximations depending on the model.

Simplification algorithm	*Running time for model of about 70 000 faces*
Hoppe [1996]	51 minutes
Guéziec [1995]	8 minutes
Lindstrom and Turk [1998]	2.5 minutes
Garland and Heckbert [1997]	15 seconds
Bi-Star Planarity	9 seconds

Table 1: Runtime comparison for different algorithms.

Models	*#v*	*#f*	*Initialization from disk*	*Rendering time*
Cessna^{NM}	3745	3946	0.280 secs	0.020 secs
Cessna^{T}	6795	13546	0.681 secs	0.030 secs
Bunny	34835	69473	3.855 secs	0.281 secs
Horse	48485	96966	5.378 secs	0.390 secs

Table 2: Initialization and rendering times.

7 Experimental results

7.1 Storage cost

According to the Euler formula for 2-dimensional triangular meshes, a mesh with n vertices has about $2n$ faces and $3n$ edges. So, assuming that floats and pointers are 4 bytes each, the runtime space cost for an AIF mesh with n vertices is as follows: $(3 \cdot 4 + 3 \cdot 4) \cdot n = 24n$ bytes for vertex coordinates and normals, $(2 \cdot 4 + 2 \cdot 4) \cdot 3n = 48n$ bytes for edges (references to vertices and faces), $(3 \cdot 4) \cdot 2n = 24n$ bytes for faces (references to edges). Thus, an AIF mesh occupies $96n$ bytes in main memory. In contrast, the overall main memory size for FastMesh [2001] and Direct Edges [1998] is $106n$ and $120n$ bytes, respectively. The conciseness of the AIF data structure comes from the fact that it does not include oriented cells (e.g. half-edges).

7.2 Time cost

The runtime performance tests have been performed on a PC equipped with 1.6GHz Intel Pentium 4, 256MB of memory, an NVidia Riva TNT2 graphics card, and running Windows 2000 OS.

Table 2 shows some results with different AIF models[1]. The AIF data structure initialization from a disk file includes:

1. Load an AIF mesh from an ASCII file describing a model.
2. Create a normal vector for each vertex of an AIF mesh for visualization.
3. Create a list of GL polygons for rendering an AIF mesh a *posteriori*.

Rendering large meshed objects such as, for example, a bunny (Figure 2(a)), a horse (Figure 2(b)), and a non-triangular, non-manifold Cessna (Figure 2(c)) is pretty fast as shown in Table 2, even considering that GL (Graphics Library) rendering is done by software instead of hardware pipeline. Possibly, the rendering time could be even shortened by using C code, instead of C++.

The great advantage of the Bi-Star Planarity simplification algorithm presented in Section 6 is that its processing time is shorter than others found in the literature [Lindstrom and Turk 1998; Cignoni et al. 1997]. This is mainly due to its geometric criterion and supporting AIF data structure. In Table 1, we can see the total

[1]Cessna^{NM} is a non-triangular, non-manifold mesh.
Cessna^{T} is a triangular manifold mesh.

processing time spent by different simplification algorithms for a model with about 70000 faces (e.g. a bunny mesh).

After running the Quadric Metrics algorithm [Garland and Heckbert 1997] and Bi-Star Planarity algorithm on the same computer, we came to the time results presented in Figures 5 and 6 for two meshes, bunny and horse, respectively. Note that the comparison between these algorithms were made for approximations with the same number of faces.

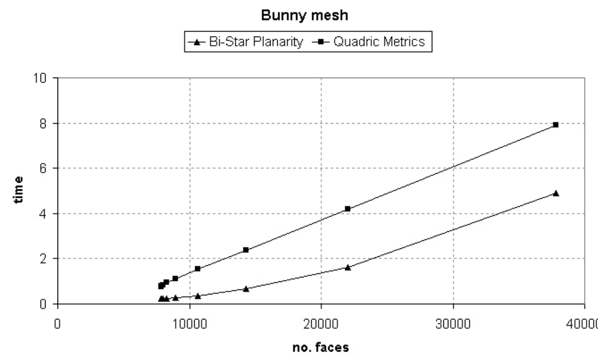

Figure 5: Simplification times for a bunny mesh.

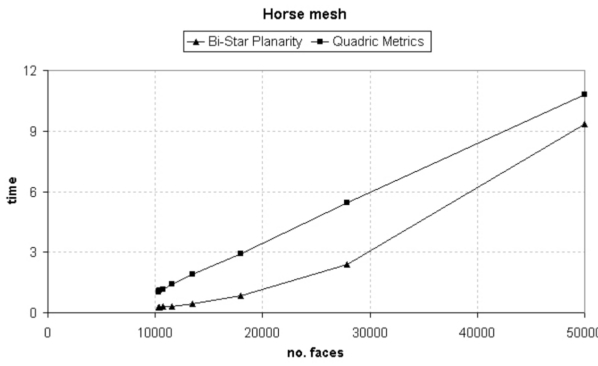

Figure 6: Simplification times for a horse mesh.

7.3 Mesh quality

The mesh quality can be evaluated by different methods (see [Zelinka and Garland 2002; Roy et al. 2002; Aspert et al. 2002; Cignoni et al. 1996] for more details). In this study, we have used the Metro geometric comparison tool to measure the mesh quality [Cignoni et al. 1996]. This tool computes the maximum and mean geometric errors of a simplified mesh with respect to the original mesh. This is illustrated in Figures 7 and 8 for two meshes, a bunny mesh and a horse mesh, respectively. The mesh quality might be improved if a point other than the midpoint of the collapsing edge were used in the Bi-Star Planarity algorithm. Such a point would be chosen in order to minimize the mesh error, but that depends on the target application.

8 Conclusions and future work

In practice, the AIF data structure and its companion mask operator proved to be a powerful tool in handling multiresolution meshes. It

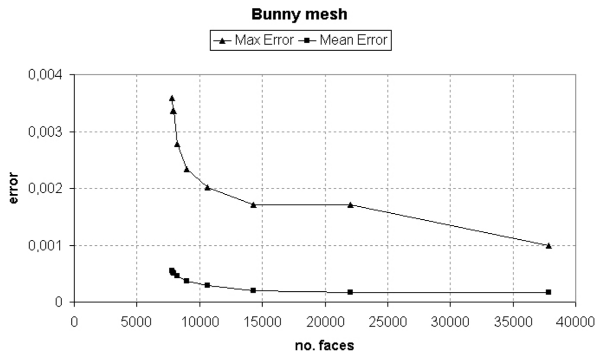

Figure 7: Max and mean error for a bunny mesh.

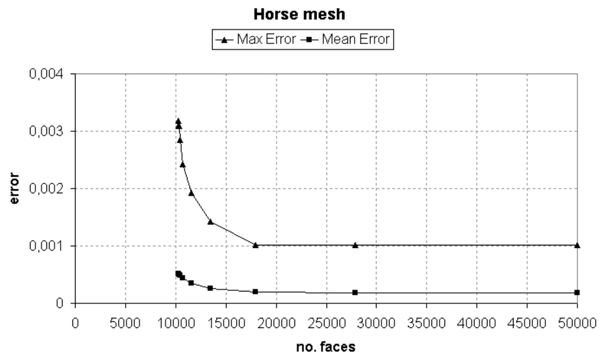

Figure 8: Max and mean error for a horse mesh.

is not necessary to be concerned about oriented cells as is usual in conventional b-reps. Recall that, in terms of the number of direct and indirect accesses, AIF query complexity presents the best performance in the class C_4^9. It is also more concise than similar data structures.

Additionally, it is able to accommodate generic meshes (cell complexes), not only triangular meshes (simplicial complexes) even in higher dimensions. Thus, it can be used in disparate applications including geometric modelling, computer aided design, and computer graphics.

On the other hand, the simplification method presented in this paper provides good results in terms of the shape preservation and time performance. This fact facilitates the development of almost real-time applications in games technology or virtual environments with adequate supporting hardware.

The performance evaluation of other simplification algorithms on the AIF data structure remains an open issue. We also believe that it is possible to improve the mesh quality by minimizing the error of simplified meshes. Besides, in the near future, we hope that the AIF data structure can address other application requirements, in particular sub-meshing for animation systems.

Acknowledgments

The horse, bunny and cessna models are courtesy of Cyberware (http://www.cyberware.com), Stanford University (http://graphics.stanford.edu), and Viewpoint Digital Inc. (http://www.viewpoint.com), respectively. A last word for the referees who have contributed to improve the paper with their valuable comments.

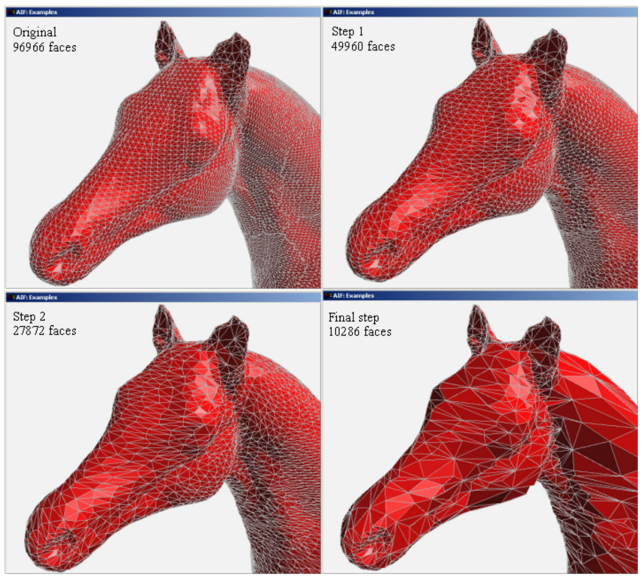

Figure 9: Results of Bi-Star Planarity algorithm for the horse.

References

ASPERT, N., SANTA-CRUZ, D., AND EBRAHIMI, T. 2002. Mesh: Measuring errors between surfaces using the hausdorff distance. In *Proc. of the IEEE International Conference in Multimedia and Expo (ICME)*, vol. 1, 705–708.

BAUMGART, B. C. 1972. Winged-edge polyhedron representation. Tech. rep., STAN-CS-320, Stanford University.

BRISSON, E. 1993. Representing geometric strutures in d dimension: topology and order. *Discrete & Computational Geometry 9*, 4, 387–426.

CAMPAGNA, S., KOBBELT, L., AND SEIDEL, H.-P. 1998. Directed edges — A scalable representation for triangle meshes. *Journal of Graphics Tools: JGT 3*, 4, 1–12.

CAREY, R., BELL, G., AND MARRIN, C. 1997. The reality modeling language iso/iec 14772-1. Tech. rep., The VRML Consortium Incorporated.

CIGNONI, P., ROCCHINI, C., AND SCOPIGNO, R. 1996. Metro: measuring error on simplified surfaces. Tech. rep., B4-01-01-96, I.E.I-C.N.R., Pisa, Italy.

CIGNONI, P., MONTANI, C., AND SCOPIGNO, R. 1997. A comparison of mesh simplification algorithms. *Computers and Graphics 22*, 1, 37–54.

COHEN, J., VARSHNEY, A., MANOCHA, D., TURK, G., WEBER, H., AGARWAL, P., FREDERICK BROOKS, AND WRIGHT, W. 1996. Simplification envelopes. In *Proc. Siggraph '96*, 119–128.

ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. 1995. Multiresolution analysis of arbitrary meshes. *Computer Graphics 29*, Annual Conference Series, 173–182.

GARLAND, M., AND HECKBERT, P. S. 1997. Surface simplification using quadric error metrics. In *Computer Graphics*, vol. 31, 209–216.

GARLAND, M. 1999. Multiresolution modeling: Survey & future opportunities. In *Eurographics '99 - State of the Art Reports*, 111–131.

GUÉZIEC, A. 1995. Surface simplification with variable tolerance. In *Second Annual Intl. Symp. on Medical Robotics and Computer Assisted Surgery (MRCAS '95)*, 132–139.

HECKBERT, P. S., AND GARLAND, M. 1997. Survey of polygonal surface simplification algorithms. In *Siggraph'97 Course Notes 25*.

HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. 1993. Mesh optimization. In *Proc. Siggraph'93*, 19–26.

HOPPE, H. 1996. Progressive meshes. In *Siggraph'96 Conference Proceedings*, 99–108.

HOPPE, H. 1998. View-dependent refinement of progressive meshes. Tech. rep., MSR-TR-98-02, Microsoft Research.

HUBELI, A., AND GROSS, M. 2000. A survey of surface representations for geometric modeling. Tech. rep., #335, Swiss Federal Institute of Technology Zurich.

KIM, J., AND LEE, S. 2001. Truly selective refinement of progressive meshes. In *Proc. of Graphics Interface 2001*, 101–110.

KLEIN, R., LIEBICH, G., AND STRASSER, W. 1996. Mesh reduction with error control. In *IEEE Visualization '96*, R. Yagel and G. M. Nielson., Eds., 311–318.

KOBBELT, L., CAMPAGNA, S., AND SEIDEL, H.-P. 1998. A general framework for mesh decimation. In *Graphics Interface*, 43–50.

LEE, A., DOBKIN, D., SWELDENS, W., AND SCHRÖDER, P. 1999. Multiresolution mesh morphing. In *Siggraph 1999, Computer Graphics Proceedings*, Addison Wesley Longman, Los Angeles, A. Rockwood, Ed., 343–350.

LINDSTROM, P., AND TURK, G. 1998. Fast and memory efficient polygonalsimplification. In *IEEE Visualization '98 Conference proceedings*, 297–286.

LOOP, C. 2000. Managing adjacency in triangular meshes. Tech. rep., No. MSR-TR-2000-24, Microsoft Research.

LOUNSBERY, M., DEROSE, T. D., AND WARREN, J. 1997. Multiresolution analysis for surfaces of arbitrary topological type. *ACM Transactions on Graphics 16*, 1, 34–73.

MANTYLA, M. 1988. *An Introduction to Solid Modeling*. Computer Science Press.

NI, X., AND BLOOR, M. S. 1994. Performance evaluation of boundary data structures. *IEEE Computer Graphics and Applications 14*, 6, 66–77.

PAJAROLA, R. 2001. Fastmesh: Efficient view-dependent meshing. In *Proceedings Pacific Graphics 2001*, 22–30.

POPOVIC, J., AND HOPPE, H. 1997. Progressive simplicial complexes. In *Computer Graphics*, vol. 31, 217–224.

ROY, M., NICOLIER, F., FOUFOU, S., TRUCHETET, F., KOSCHAN, A., AND ABIDI, M. 2002. Assessment of mesh simplification algorithm quality. In *Proceedings of SPIE Electronic Imaging*, vol. 4661, 128–137.

SCHROEDER, W. J., ZARGE, J. A., AND LORENSEN, W. E. 1992. Decimation of triangle meshes. In *Proc. Siggraph'92*, 65–70.

TAUBIN, G., AND ROSSIGNAC, J. 1998. Geometric compression through topological surgery. *ACM Transactions on Graphics 17*, 2, 84–115.

TOUMA, C., AND GOTSMAN, C. 1998. Triangle mesh compression. In *Proceedings of Graphics Interface'98*, Morgan Haufmann Publishers, K. B. W. Davis and A. Fournier, Eds., 26–34.

TURK, G. 1992. Re-tiling polygonal surfaces. *Computer Graphics 26*, 2, 55–64.

WAVEFRONT TECHNOLOGIES, I. 1998. Wavefront file formats. Tech. rep., Version 4.0 RG-10-004 (1ł ed.), Santa Barbara, California, USA.

WEILER, K. 1985. Edge-based data structure for solid modelling in curved-surface environments. *IEEE Computer Graphics and Applications 5*, 1, 21–40.

WEILER, K. 1998. The radial edge struture: a topological representation for non-manifold geometric boundary modelling. *Geometric Modelling for CAD applications*.

ZELINKA, S., AND GARLAND, M. 2002. Permission grids: Pratical, error-bounded simplification. *ACM Transaction on Graphics 21*, 2, 207–229.

ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. 1997. Interactive multiresolution mesh editing. In *SIGGRAPH 97 Conference Proceedings*, 259–268.