Compression-Domain Editing of 3D Models

Tulika Mitra
School of Computing
National University of Singapore
tulika@comp.nus.edu.sg

Tzi-cker Chiueh
Department of Computer Science
SUNY at Stony Brook
chiueh@cs.sunysb.edu

Abstract

3D models have become an essential element of multimedia applications because they provide visual effects that permit interactive exploration. As in other media types, efficient compression of 3D models is essential to reduce the associated storage and processing cost. Triangle mesh is the prevailing geometric representation for 3D scene models. Despite significant research on compression algorithms for triangle mesh in recent years, most 3D model editing tools still manipulate triangle meshes in their uncompressed representation. This paper presents a novel *compression-domain mesh editing* (CDE) technique, which supports an efficient lossless mesh compression algorithm called BFT and allows 3D models to be directly edited based on the BFT-compression form. Experimental results from real-world complex 3D models running on a fully operational CDE prototype demonstrate that compared to the editing of uncompressed triangle mesh, the CDE technique achieves a reduction in run-time memory requirements during the editing process by a factor of 14, while keeping the average edit operation latency under 2 msec regardless of the size of the 3D models.

1 Introduction

Triangle mesh, a connected set of triangles, is the prevailing representation of 3D graphics models. The increasing complexity of 3D graphics models translates to escalating mesh size, which often times could overwhelm the underlying hardware's capability. The general solution to this problem is to compress 3D models as much as possible and to keep them in the compressed form for as long as possible. In recent years, there has been a considerable amount of research on fast and efficient compression/decompression algorithms for triangle mesh. However, the focus of these research has invariably been on minimizing the size of the compressed triangle mesh only. The issue of keeping triangle mesh in the compressed form as long as possible is largely left unexplored. In many cases, when applications such as authoring or simplification tools manipulate triangle mesh, it is desirable to operate directly on their compressed representation to reduce the total memory footprint. However, to the best of our knowledge, all existing triangle mesh editing tools available today work on triangle mesh representations that maintain explicit information about the vertices, edges, and triangles. Such representations are usually too memory-intensive, resulting in virtual memory thrashing and sometimes even system crash.

In this paper, we present a novel compression-domain triangle mesh editing technique (CDE) that is able to keep triangle mesh-based 3D models in the compressed form across various types of editing operations. CDE allows editing operations to be performed directly on a compressed mesh representation. More concretely, a CDE editor takes as input a compressed mesh representation and the requested editing operations, and generates the edited version of the compressed mesh *without explicit decompression/recompression of the entire mesh*. CDE provides two major advantages: (1) the editing tool does not need to maintain a decompressed representation in memory, thereby reducing the memory consumption in the editing process and (2) it improves editing interactivity since a compressed mesh can be rendered faster than its uncompressed counterpart [11] by exploiting the vertex/edge sharing property of triangle meshes.

2 Related Work

There has been a considerable amount of research on compressing the topological information of triangle mesh. The pioneering work on mesh compression was done by Deering [3]. This work has been followed by several more efficient triangle mesh representations [2, 13, 7, 10, 12, 14]. In triangle mesh editing, previous research has mostly focused on developing data structures for easier manipulation of topological information in the triangle mesh such as Winged-Edge [1], Split-Edge [4], and Hybrid-Edge [8] model. Hierarchical/multi-resolution editing allows an user to modify a model at different resolutions. The different approaches of hierarchical editing include H-splines [6], wavelet representations [5], and subdivision surfaces [15]. The CDE technique proposed in this paper is the first known compression-domain processing example for 3D models in general and for triangle mesh in particular.

3 Triangle Mesh Compression

In this section, we briefly describe the *Breadth-First Traversal (BFT)* [10, 9] algorithm for triangle mesh compression. A triangle mesh is represented with *geometry* (a set of vertex positions, color, and other attributes) and *connectivity* (the incidence relations among vertices, edges, and triangles). Traditionally, each triangle in a triangle mesh is represented independently in terms of the geometry of its three vertices. Triangle mesh compression consists of (1) lossless connectivity compression and (2) lossy geometry compression. BFT is a connectivity compression algorithm. However, any efficient geometry compression algorithm can be trivially integrated with the BFT algorithm.

The basic idea of the BFT algorithm is to traverse a triangle mesh in a breadth-first order from a *seed triangle*. The vertices of the seed triangle form a *frontier*. A frontier is a circular buffer of vertices. BFT visits each edge — consisting of two consecutive vertices — of the frontier and enumerates the unvisited triangle, if any, that is incident on that edge by specifying its *third vertex*. At the same time, it incrementally modifies the frontier to delete the vertices whose incident triangles have all been visited, and to add the new vertices. BFT continues to enumerate the triangles and modify the frontier till either

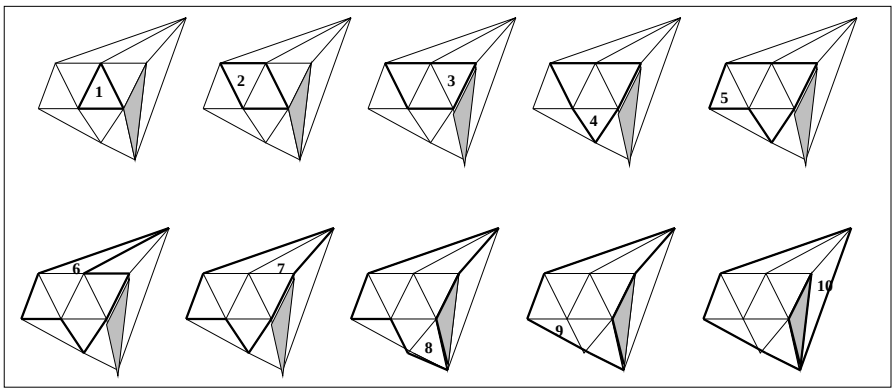

Figure 1: *BFT compression algorithm*

there is only one vertex left in the frontier, or a frontier left with n vertices has not been modified for n consecutive steps. Figure 1 illustrates this traversal process with a small triangle mesh. The shaded portion in the figure is a hole in the triangle mesh. The bold lines indicate the frontier. The ordering of the triangles represents the order in which the triangles are enumerated during the BFT traversal.

The edge for which BFT attempts to find an incident and not-yet-visited triangle is called the *current edge*. A current edge for which BFT cannot find any unvisited triangle, because either it is a boundary edge or both of its incident triangles have been visited, is called a *null edge*. The third vertex used to form a triangle with the current edge can be represented either explicitly in terms of its geometric coordinates or implicitly as a pointer to some vertex in the frontier, specified as an offset from one of the current edge vertices.

Given an input triangle mesh, BFT performs the following two steps: (1) it pre-processes the triangle mesh to find out the visiting order of the triangles; and (2) it represents the mesh as a variable-length command sequence, where each command encodes either a new triangle in terms of the corresponding third vertex or the presence of a null edge. We store geometry data for all the vertices separately as a vertex array sorted in the order in which they *first* appear in the BFT mesh. The BFT decompression algorithm dynamically reconstructs the frontier of the BFT traversal and enumerates triangles on the frontier according to the information encoded in the command sequence.

4 Compression-Domain Mesh Editing

When a 3D mesh is imported for editing, the CDE editor first builds meta-data to facilitate the mapping between user-clicked regions and the underlying geometric objects. Then it allows users to interactively edit the compressed mesh and visualize the corresponding rendered image until the result is satisfactory. Periodically the system also needs to recompress the edited mesh to make up for the compression efficiency lost during editing.

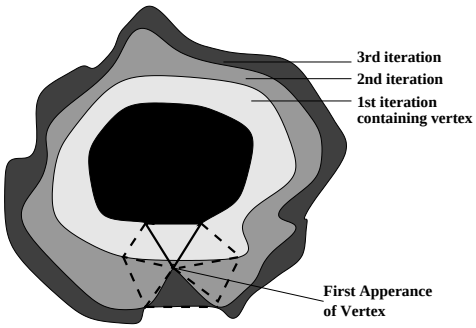

Figure 2: *Distance between a vertex and its incident edges and faces in a BFT mesh.*

4.1 Mesh Edit Primitives

CDE supports a set of fundamental edit primitives that are representative of those that standard 3D model editors support: (1) *move-vertex(v):* change the coordinate of v (2) *delete-vertex(v):* delete v and its incident faces, (3) *delete-edge(e):* delete e and its incident faces, (4) *delete-face(f)*: delete face f but do not delete the constituent edges of f, and (5) *add-face(v1, v2, v3):* add a new face consisting of vertices v1, v2, and v3.

The success of a CDE editor is dependent on an efficient implementation of each fundamental edit primitive in compression domain. Any efficient implementation should be able to (1) quickly find the portion of the BFT mesh that needs to be modified, and (2) locally modify/edit that portion to generate the BFT representation of the resulting mesh.

4.2 Where to Edit

The portion of the BFT mesh where an edit primitive operates is called the *target mesh point*. A naive way to implement an edit operation is to decompress the input BFT mesh up to the target mesh point, and then make the necessary modifications. But for each edit primitive, this implementation takes time proportional to the distance of the target mesh point from the beginning of the BFT mesh which may become prohibitively slow if the target mesh point corresponds to the end of the BFT command stream. To address this problem, CDE uses a data structure called the *location map*, which maps an edit primitive to a position close to the corresponding target mesh point. The position returned by the location map is the *source mesh point*. Given an edit operation, CDE quickly homes in to the target mesh point by performing *incremental decompression* of the edited mesh from the source mesh point.

Location Map Because the BFT compression algorithm performs a breadth-first traversal of the input mesh, *the BFT command that enumerates an edge or a face that contains a particular vertex appears at most a few iterations away from the first appearance of that vertex in the command stream*. Figure 2 illustrates this observation. To identify the first appearance of a vertex, CDE divides a BFT command stream into multiple non-overlapping command substreams. The location map is an array with each entry corresponding to a command substream. Figure 3 shows the location map's structure. Each entry in the location map contains three fields: (a) Location: location of the first command in the sub-

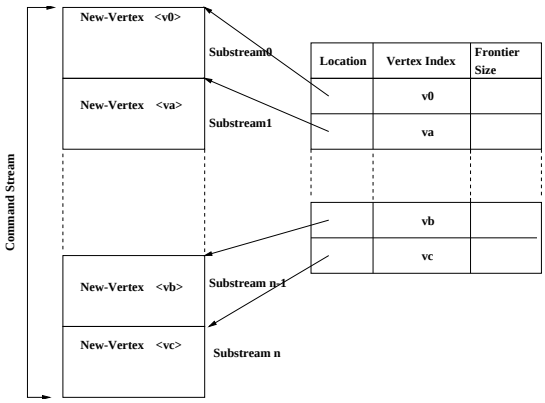

Figure 3: *Location Map meta-data to correlate vertices and their positions in BFT mesh.*

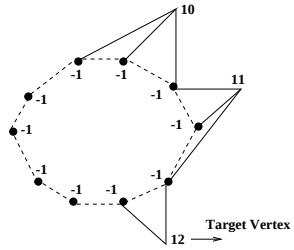

Figure 4: *Incremental decompression: The dotted lines indicate the dummy frontier with 10 vertices. All new vertices encountered in incremental decompression have valid attributes.*

stream, (b) Frontier-size: size of the frontier immediately before the first command in the substream, and (3) Vertex-index: index of the first vertex that is explicitly represented in the substream.

Note that the vertex array is sorted in the order in which the vertices appear in the command stream. The location map is sorted with respect to the location field; so it is also sorted with respect to the vertex-index field. Given a vertex index, CDE performs a binary search through the location map to identify the command substream that contains the first appearance of the vertex. For a vertex edit operation, CDE looks up the location map with the vertex index. For an edge (face) edit operation, BFT uses the smallest vertex index of the edge (face) as the input to location map lookup.

Incremental Decompression Decompression of a BFT command depends on all the previous commands in the stream. However, with the help of the *frontier-size* field in the location map, it is possible to restrict decompression to the command substream that contains the BFT command in question. The idea is to form a *dummy frontier* containing as many vertices as indicated by the frontier-size field, and to start decompression based on this dummy frontier. Each of the vertices in this dummy frontier is unknown and is represented by -1. Fortunately, CDE does not need to know the exact vertices on the dummy frontier because the edit primitive by construction cannot be edges/faces that involve dummy frontier vertices. As the traversal progresses, eventually all the vertices associated with the edit primitive in question will appear in the frontier, as illustrated in Figure 4. This incremental decompression continues till the desired vertex, edge, or face in encountered.

Dataset	Vertex	Triangle	Edge
Skull	20,002	40,000	60,000
Bunny	34,834	69,451	104,288
Horse	48,485	96,966	145,449
Hand	327,323	654,666	981,999
Dragon	437,645	871,414	1,309,256
Buddha	543,652	1,087,716	1,631,574
Blade	882,954	1,765,388	2,648,082

Table 1: *The characteristics of the triangle mesh models used in our study.*

4.3 How to Edit

The semantic of the three delete primitives requires only the deletion of one or more faces. It is possible to replace the command that enumerates a face with a different command so as to delete the face. For CDE, however, there is a problem with this approach. BFT commands use variable-length encoding. Therefore, if CDE changes a command in the middle of the command stream, it has to shift all the following commands — an operation that incurs significant I/O overhead. To support local editing of a BFT mesh, we add an extra *status bit* to all the BFT commands that enumerate a face. CDE initializes all the status bits to 0 before the editing session starts. As faces are deleted, CDE changes the corresponding status bits to 1. For software rendering, if the status bit is 0, the decompressor sends the corresponding triangle to the rendering pipeline; otherwise it does not send the triangle. In addition, a flag at the beginning of the BFT mesh indicates whether the command stream includes status bits. The status bits add 1-bit overhead per triangle to the BFT mesh *during editing*. But CDE removes the status bits from the final BFT mesh after all the edit operations for that mesh are complete.

The last edit primitive adds a face to the BFT mesh. To avoid modification in the middle of the BFT mesh, we append a component mesh containing only that face to the BFT representation. But the vertices of the face may already be present in the BFT representation leading to multiple explicit representations. To avoid inconsistency, CDE maintains a *vertex instance* table to quickly find all the explicit vertex instances. The table, sorted with respect to the vertex index, contains pointers to the multiple explicit representations of the vertex in the BFT mesh. As CDE performs recompression of the edited mesh periodically, the vertex instance table cannot grow to infinite size.

5 Performance Evaluation

Based on a prototype implementation of the proposed compression-domain triangle mesh editor, we compare the peak memory requirement and edit performance of the CDE editor and a traditional mesh editor. The 3D models used in this study and their characteristics are shown in Table 1. All measurements are taken on a Pentium II 300-MHz machine with 320 MBytes of memory running Linux.

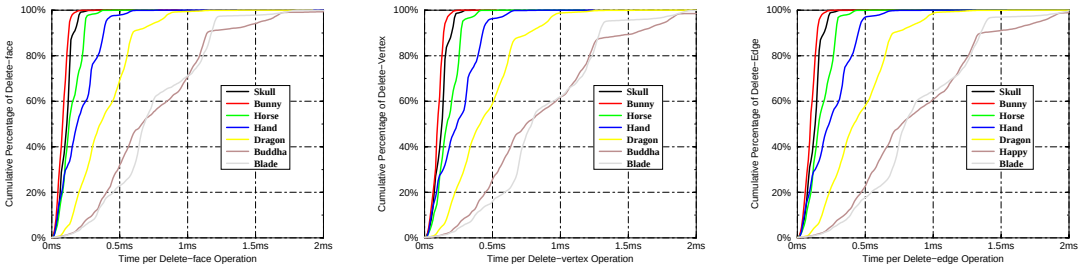

Figure 5: *Distribution of the time required for the edit primitives.*

Dataset	Additional Data-Structure			Peak Memory		
	Winged Edge	Adj.	CDE (% of Adj.)	Winged Edge	Adj.	CDE (% of Adj.)
Skull	2,160	640	9(1.4%)	2,800	1,280	190(14.8%)
Bunny	4,312	1,112	13(1.1%)	5,424	2,224	325(14.6%)
Horse	5,236	1,551	16(1.0%)	6,788	3,103	453(14.6%)
Hand	40,589	10,475	89(0.8%)	51,064	20,949	3,033(14.5%)
Dragon	54,078	13,958	128(0.9%)	68,036	27,916	4,108(14.7%)
Buddha	67,434	17,402	155(0.9%)	84,836	34,803	5,114(14.7%)
Blade	109,118	28,248	256(0.9%)	137,366	56,497	8,236(14.6%)

Table 2: *Additional data-structure size and maximum memory requirement for winged-edge, adjacency, and CDE (in KBytes)*

5.1 Memory Requirement

The key advantage of the CDE editor is its smaller memory requirement compared to traditional 3D model editors based on an uncompressed mesh representation. Traditional editors require additional data structures to quickly retrieve the adjacency relations. We choose the *winged-edge* data structure [1] because other data structures are just variants of winged-edge. Winged-edge requires large memory footprint because it is meant to handle polygonal as well as triangle mesh. For a fair comparison with CDE, we use another data structure for traditional editors, called *adjacency*, which maintains pointers from each vertex to all the incident faces. Adjacency works only for triangle mesh, but it entails smaller memory footprint requirement and can easily retrieve adjacency relations.

First, we compare the memory requirements for auxiliary data structures between the CDE editor and traditional editors. The auxiliary data structures of the CDE editor include the the location map, the vertex instance table, and the maximum size of frontier memory. In the CDE prototype, we use one location map entry per 100 BFT commands. Table 2 shows that the CDE editor's memory requirement for auxiliary data structure is under 1.4% of that of a traditional mesh editor using adjacency data structure, and only 0.25% of that of winged-edge based editors. The peak memory requirement of a 3D model editor includes, in addition to auxiliary data structures, the connectivity and the geometry of the triangle mesh being edited. Both the adjacency editor and winged-edge based editor use the indexed independent triangle representation. CDE, on the other hand, uses the BFT representation whose connectivity cost is only 2-3% of that of the uncompressed mesh rep-

Dataset	Delete-face		Delete-vertex		Delete-edge	
	CDE	GEN	CDE	GEN	CDE	GEN
Skull	113	2.89	130	2.32	133	3.11
Bunny	88	2.99	105	2.32	106	3.23
Horse	152	2.97	183	2.32	183	3.33
Hand	215	3.19	253	2.33	256	3.31
Dragon	397	3.22	453	2.33	462	3.43
Buddha	758	3.22	866	2.33	881	3.44
Blade	758	3.20	881	2.33	885	3.32

Table 3: *Average time (μsec) required for an edit primitive. GEN stands for Generic.*

resentation [9]. We use 8 bytes for per-vertex geometry information: 16-bit coordinates and 15 bit color values [3]. Table 2 shows that CDE needs about 14% of the memory required by an adjacency-based editor, and 7% of that of a winged-edge based editor.

5.2 Edit Operation Performance

We compare the edit performance between the prototype CDE system and a generic 3D model editor that uses the adjacency data structure. For small models the CDE editor allows interactive editing although it is slower than the generic editor, but for large 3D models, the CDE editor still allows interactive editing whereas the generic editor is completely halted. A 3D model is large if it leads to excessive paging at run time.

Table 3 compares the edit operation performance between the CDE and the generic editor in terms of the *average* time required to perform the three edit primitives: delete-face, delete-vertex, and delete-edge. The other two edit primitives, move-vertex and add-face, do not involve incremental decompression and thus require almost the same time in CDE as in the generic editor. We measure the time taken by an edit primitive when it is applied to *each* face, vertex, and edge in the input triangle mesh and report their average. Even though the average edit operation latency of the CDE editor is longer than that of the generic editor, it is less than 1 msec, which is quite reasonable for interactive editing. Figure 5 shows the distribution of latency for the three edit primitives in the CDE editor. The X axis represents the time required for a single edit primitive. The Y axis represents the cumulative percentage of edit primitives whose measured latency is smaller than the amount of time represented on the X axis. For all the test triangle meshes, 99.9% of the edit primitives can be performed within 2 msec.

An important optimization that the CDE editor can incorporate is to perform a sequence of edit primitives in a batch. For example, if a user deletes a small portion of the triangle mesh, the CDE editor can perform *only one* incremental decompression to delete all the faces, thus amortizing the per-edit incremental decompression overhead over a larger number of edit primitives and decreasing the average edit primitive latency. Figure 6 shows how batching of delete-vertex primitives decreases the per-operation latency by more than 50%. We believe that this approach can further bring closer the gap between the edit primitive latencies of the CDE and the generic editor.

To emulate paging effects of large 3D models, we restrict the physical memory of the

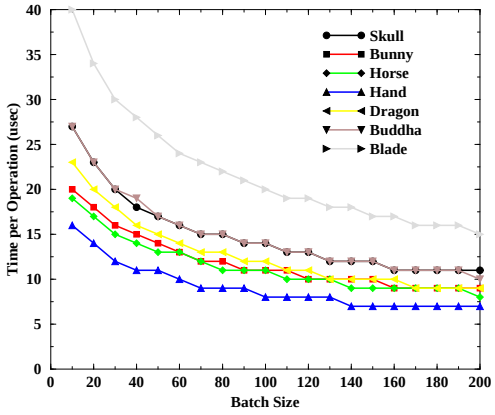

Figure 6: *Batching of delete-vertex primitives. X-axis shows the number of delete-vertex primitives in a batch. Y-axis shows per-operation latency in μsec. We choose 1000 random regions for each batch size and report the average edit time.*

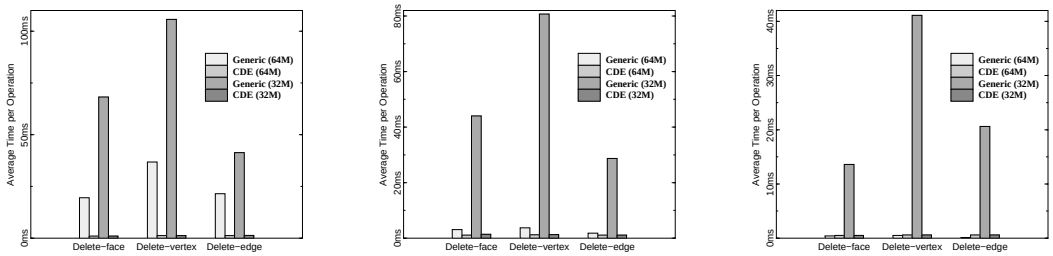

Figure 7: *The paging effect for large triangle mesh: Blade, Buddha, and Dragon.*

experimental machine to 64MB and 32MB. We perform the same set of face, vertex, and edge delete operations using CDE and the generic editor for three largest triangle meshes: "Dragon", "Buddha", and "Blade". Figure 7 shows the average time required for edit primitives for two memory sizes. With 32MB memory, the CDE editor is significantly faster than the generic editor for all the three triangle meshes. The CDE editor is able to keep its memory footprint smaller than 32MB, and thus avoids almost all the paging cost. With 64MB of memory, the average edit operation latency of the CDE editor is still less than the generic editor for "Blade" and "Buddha", and is comparable to the generic editor for "Dragon." As expected, the larger the 3D model, the more the performance improvement of the CDE editor because of the reduction in paging overhead.

Overall the CDE editor is more scalable than the generic editor because its performance is more stable with increasing 3D model size. The average edit operation latency of the CDE editor increases from 1 msec to 1.5 msec as the memory size is decreased from 320MB to 32MB. For the same change in memory size, the average edit latency of the genetic editor increases from 3 μsec to 100 msec, a 4 to 5 orders of magnitude difference!

6 Conclusion

We present the design, implementation, and evaluation of a novel compression-domain triangle mesh editing technique. The CDE technique, based on an efficient lossless mesh compression algorithm called BFT, allows common triangle mesh editing operations to be performed directly on compressed triangle meshes. The CDE prototype implementation is able to achieve a significant reduction in memory requirement during the mesh editing process without incurring noticeable computation overhead. Given the increasing importance of 3D graphics in Internet multimedia applications, the CDE technology is expected to have a significant impact on the future development of media content authoring systems.

7 Acknowledgment

This research was supported by the following grants: NSF MIP-9710622, NSF ACI-9907485, and a grant from Sandia National Laboratory.

References

[1] B. Baumgart. Winged Edge Polyhedron Representation. Technical Report CS-320, Stanford Artificial Intelligence Laboratory, 1972.

[2] Mike. M. Chow. Optimized Geometry Compression for Real-time Rendering. In *Proceedings of the 8th Annual IEEE Visualization Conference*, 1997.

[3] M. Deering. Geometry Compression. In *Proceedings of SIGGRAPH*, 1995.

[4] C. M. Eastman. *Introduction to Computer-Aided Design*, 1982. Course Notes.

[5] A. Finkelstein and D. H. Salesin. Multiresolution Curves. In *Proceedings of the Annual ACM Conference on Computer Graphics (SIGGRAPH)*, 1994.

[6] D. R. Forsey and R. H. Bartels. Hierarchical B-Splines Refinement. In *Proceedings of the Annual ACM Conference on Computer Graphics (SIGGRAPH)*, 1988.

[7] S. Gumhold and W. Straβer. Real Time Compression of Triangle Mesh Connectivity. In *Proceedings of SIGGRAPH*, 1998.

[8] Y. E. Kalay. The Hybrid Edge: A Topological Data-Structure for Vertically Integrated Geometric Modelling. Technical report, SUNY at Buffalo, 1988.

[9] T. Mitra. *Mesh Compression and Its Hardware/Software Applications*. PhD thesis, Department of Computer Science, SUNY at Stony Brook, 2000.

[10] T. Mitra and T. Chiueh. A Breadth-First Approach to Efficient Mesh Traversal. In *Proceedings of the 13th ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware*, 1998.

[11] T. Mitra and T. Chiueh. An FPGA Implementation of Triangle Mesh Decompression. In *Proceedings of the IEEE International Symposium on Field Programmable Custom Computing Machines*, 2002.

[12] J. Rossignac. Edgebreaker: Connectivity Compression for Triangle Meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1), 1999.

[13] G. Taubin and J. Rossignac. Geometry Compression through Topological Surgery. *ACM Transaction on Graphics*, 17(2), 1998.

[14] C. Tauma and C. Gotsman. Triangle Mesh Compression. In *Proceedings of the 24th Annual Graphics Interface Conference(GI)*, 1998.

[15] D. Zorin, P. Schröder, and W. Sweldens. Interactive Multiresolution Mesh Editing. In *Proceedings of the 24th Annual ACM Conference on Computer Graphics (SIGGRAPH)*, 1997.