GEOMETRY COMPRESSION OF TETRAHEDRAL MESHES USING OPTIMIZED PREDICTION

Dan Chen, Yi-Jen Chiang, Nasir Memon, and Xiaolin Wu

Department of Computer and Information Science
Polytechnic University, Brooklyn, NY 11201, USA
dchen01@utopia.poly.edu, {yjc, memon, xwu}@poly.edu

ABSTRACT

In this paper we propose a novel geometry compression technique for volumetric datasets represented as tetrahedral meshes. We focus on a commonly used technique for predicting vertex geometries via a flipping operation using an extension of the parallelogram rule. We demonstrate that the efficiency of the flipping operation is dependent on the order in which tetrahedra are traversed and vertices are predicted accordingly. We formulate the problem of optimally (traversing tetrahedra and) predicting the vertices via flippings as a combinatorial optimization problem of constructing a *constrained minimum spanning tree*. We give heuristic solutions for this problem and show that we can achieve prediction efficiency very close to that of the *unconstrained* minimum spanning tree which is an unachievable lower bound. We also show significant improvements of our new geometry compression over the state-of-the-art flipping approach, whose traversal order does not take into account the geometry of the mesh.

1. INTRODUCTION

In the past several years, new challenges for scientific visualization have emerged as the size of data generated from the simulations has grown exponentially. The emerging demand for efficiently storing, transmitting, and visualizing such data in networked environments has motivated graphics compression for 3D polygonal models and volumetric datasets to become a focus of research in the past several years. For volumetric data, the most general class is *irregular-grid* volume data represented as a *tetrahedral mesh*. It has been proposed as an effective means of representing disparate field data that arise in a broad spectrum of scientific applications including structural mechanics, computational fluid dynamics, partial differential equation solvers, and shock physics. A tetrahedral-mesh dataset consists of the following two components: *geometry*—the 3D coordinates and the *data-attribute* information (such as *scalar values* in our case) of the mesh vertices, and *connectivity*—the incidence information specifying the edges, triangle faces, and tetrahedral cells connecting the mesh vertices.

Although there has been a significant amount of research done on graphics compression, most techniques reported in the literature have mainly focused on compressing the *connectivity*, rather than the *geometry* information. As a results, while connectivity compression already achieves an impressive compression rate of 1–2 bits per triangle for triangle meshes [23, 19, 1, 24] and 2.04–2.31 bits per tetrahedron for tetrahedral meshes [8, 25], progress made in compressing the geometry information has not been equally impressive. For a tetrahedral mesh, typically each floating-point coordinate is first quantized to a 16-bit integer (thus 48 bits per vertex for the raw information, *not* including the scalar values), and about 30 bits per vertex (*not* including the scalar values) are required after compression [8]. Given that the number of tetrahedral cells is about 4.5 times the number of vertices in typical tetrahedral meshes and that the connectivity compression ratio is about 2 bits per tetrahedron, it is clear that geometry compression is by far the dominating bottleneck that needs to be worked on if we hope to obtain a significant improvement in the overall compression efficiency.

In this paper, we propose a novel geometry compression technique for tetrahedral volume data. Our approach is based on an extension of the *flipping* algorithm first introduced in [24], which, originally working for 3D triangle meshes, traverses the triangles and predicts the position of the next vertex via a *flipping* operation using the *parallelogram rule*. This flipping algorithm [24] has been dominant and widely considered as state of the art for geometry compression, and has been adopted for the MPEG-4 standard for mesh geometry coding [21]. The flipping algorithm can be extended to work for tetrahedral meshes in a similar manner, traversing the tetrahedra and predicting the position (as well as the scalar value) of the next vertex via a flipping operation, now instead of flipping across the center of the common edge shared by the current and the new triangles, flipping across the center of the common *triangle face* shared by the current and the new tetrahedra. This flipping extension, combined with the best connectivity coder [8, 25], is also considered as state of the art for geometry compression of tetrahedral meshes. However, such encoding is dominated by the connectivity coder, and traverses tetrahedra in an order that ignores the geometry of the mesh, resulting in a geometry compression ratio that is far below optimal as we show in this paper.

The main idea behind our technique is that the efficiency of the flipping operation is dependent on the order in which tetrahedra are traversed and vertices are predicted accordingly. We formulate the problem of optimally (traversing tetrahedra and) predicting the vertices via flippings as a combinatorial optimization problem of constructing a *constrained minimum spanning tree*. We give heuristic solutions for this problem and show that we can achieve prediction efficiency within 5.85% on an average compared to that of

Research of the first author is supported by NSF Grant ACI-0118915; research of the second author is supported in part by NSF Grant ACI-0118915, NSF CAREER Grant CCR-0093373, and NSF ITR Grant CCR-0081964; research of the third author is supported in part by NSF Grant ACI-0118915 and NSF Grant CCR-0208678; research of the fourth author is supported in part by NSF Grant CCR-0208678. The contact author is Yi-Jen Chiang.

the *unconstrained* minimum spanning tree which is an unachievable lower bound. We also show significant improvements (up to 30.2%) of our new geometry compression over the state-of-the-art flipping approach, whose traversal order is solely decided by the connectivity coder.

2. PREVIOUS WORK

There has been a significant amount of work on compressing polygonal meshes [5, 23, 9, 19, 14, 1, 10, 15, 17, 24]. Much of this work has mainly focussed on compressing *connectivity* information. Compression techniques for polyhedral volume meshes have also been widely studied [22, 8, 18, 25]; again the main focus has been on connectivity compression. As mentioned before, these techniques achieve an impressive compression performance of 1–2 bits per triangle for triangle meshes [23, 19, 1], and 2.04–2.31 bits per tetrahedron for tetrahedral meshes [8, 25].

There are relatively few results that focus on compressing the geometry information. Devillers and Gandoin [6, 7] have proposed compression techniques that are driven by the geometry information, for both triangle meshes and tetrahedral meshes. They only consider compression of vertex coordinates but not the associated scalar values for the case of tetrahedral meshes. Lee *et. al.* [17] proposed the *angle analyzer* approach for traversing and encoding polygonal meshes consisting of triangles and quads.

The most popular technique for geometry compression of polygonal meshes is the *flipping* algorithm based on the *parallelogram rule* first introduced by Touma and Gotsman [24] as mentioned before. Isenburg and Alliez [12] extended the parallelogram rule so that it works well for polygonal surfaces beyond triangle meshes. Isenburg and Gumhold [13] applied the parallelogram rule in their out-of-core compression algorithm for polygonal meshes larger than can fit in main memory. Other extensions of the flipping approach for compressing polygonal surfaces include the work given in [16, 3, 2].

For volume compression, Isenburg and Alliez [11] extended the flipping idea to hexahedral volume meshes. As mentioned in Section 1, the basic flipping approach of [24] can be extended to tetrahedral meshes as well, which, combined with the best connectivity coder [8, 25], is considered as state-of-the-art geometry compression technique for tetrahedral meshes. We show in Section 4 that our new geometry compression achieves improvements of up to 30.2% over this approach.

There are other compression techniques for *regular-grid* volume data (e.g., [20] and the references therein). All these approaches do not consider compressing the vertex coordinates, since such information is not needed for regular grids.

3. OUR APPROACH

3.1 Problem Formulation and Algorithm Overview

As mentioned before, our approach is based on an extension of the flipping algorithm [24] from 3D triangle meshes to tetrahedral meshes: we traverse the tetrahedra and predict the position (as well as the scalar value) of the next vertex via a flipping operation, which flips across the center of the common *triangle face* shared by the current and the new tetrahedra. Specifically, suppose (x,a,b,c) and (y,a,b,c) are two face-adjacent tetrahedra sharing a common triangle $\triangle abc$, and d is the center of $\triangle abc$ given by $d=(a+b+c)/3$. We predict the position (and scalar value) of y from x as y', where the vector (x,d) equals to the vector (d,y'), i.e., $d-x=y'-d$ (and thus $y'=2d-x$), with the *prediction error* $y-y'$. Similarly, we can predict x from y as x', where $x'=2d-y$, with the prediction error $x-x'$. We call x and y an *antipodal pair* since they can predict each other by flipping across the same face. Also, the prediction errors $y-y'$ and $x-x'$ are the same (both equal to $x+y-2d$).

Our main idea is built on the observation that there are many ways to predict a vertex v: essentially, each tetrahedron having v as a vertex gives rise to a distinct vertex w such that w and v are an antipodal pair (and w gives a distinct prediction for v). For the purpose of geometry compression, our goal is to predict each vertex *once* and to minimize the total prediction error for the whole mesh. This can be formulated as a combinatorial optimization problem, namely, the problem of finding a *minimum spanning tree (MST)* on a graph $G=(V,E)$ where the vertex set V consists of the vertices of the tetrahedral mesh, and the edge set E consists of edges (x,y) for each antipodal pair x and y. Since the prediction errors $x-x'$ and $y-y'$ are the same, each edge $(x,y)\in E$ is *undirected* with edge cost the prediction error $x-x'$ (thus G is an *undirected*, weighted graph).

This MST problem is not yet the corrected formulation of the problem, however. Notice that in order to flip from a vertex x to predict its antipodal vertex y across the common triangle face $\triangle abc$, the vertices a,b,c, as well as x, need to be known (visited) first. This is a causality constraint imposed by the flipping operation. Hence we introduce the notion of a *constrained* MST (*CMST*) which admits a traversal that does not violate the causality constraint. Our final formulation of the problem, then, is to find a CMST on G, where each edge $(x,y)\in G$ connecting antipodal vertices x and y is associated with *constraint vertices* a,b, and c that define the common triangle face $\triangle abc$ to be flipped across.

To solve the CMST problem on G, we propose a heuristic algorithm to compute an (approximate) CMST that is a feasible solution but the cost may not be optimal. Our next task is to use the constructed CMST T to traverse the tetrahedra and predict vertices accordingly, in an order satisfying the causality constraint of the flipping operations, to perform the geometry compression. Ideally, we would like to traverse the tree T on the fly while T is being constructed, but it turns out that such traversal order is very expensive to encode. Therefore, we first complete the construction of T, and then traverse T in a separate pass, in an order that still satisfies the causality constraint and yet is much cheaper to encode. Such traversal results in a *pseudo-CMST*, which in fact is a forest and may have a slightly higher prediction cost than T due to the starting cost of each tree in the forest. The *overall encoding cost*, including the starting costs and the prediction costs, however, is greatly reduced in the pseudo-CMST compared with the overall encoding cost needed for T. Finally, the geometry coder does not visit all tetrahedra in the mesh, since each vertex is spanned exactly once, where spanning a vertex corresponds to visiting a new tetrahedron, and typically the number of tetrahedra is about 4.5 times the number of vertices in a tetrahedral mesh. We still need to "collect" the left-over tetrahedra for the purpose of connectivity compression. We develop an algorithm to build a pseudo-CMST and collect the left-over tetrahedra in the same pass, in an interleaving manner; this completes both geometry and con-

nectivity encoding at the same time. In summary, our overall algorithm performs the following steps:

1. Form the graph G.
2. Construct an approximate CMST T on G.
3. Traverse T in another pass, build a pseudo-CMST and collect the left-over tetrahedra, and complete both geometry and connectivity encoding.

In the following, we describe the details of Steps 2 and 3.

3.2 Heuristic Algorithm for CMST

Given the graph $G = (V,E)$ where each edge $(x,y) \in E$ connecting antipodal vertices x and y is associated with the constraint vertices a,b,c defining the common triangle face $\triangle abc$ to be flipped across, our heuristic algorithm to build an approximate CMST T on G is a *modified* Prim's algorithm [4]. Initially, for each edge (x,y) with constraint vertices a,b, and c, we establish a *bidirectional* link between (x,y) and each of a,b, and c, so that from each edge we know its constraint vertices, and from each vertex we know the edges constrained by this vertex.

We start constructing T by including the four vertices of a tetrahedron into T, which enables initial predictions. Throughout the process, we use a priority queue Q to maintain the vertices not yet added to T. The key of each vertex x in Q is the minimum cost among the costs of the *valid* edges $(x,y) \in G$ through which x can be added to T, where (x,y) is valid if its *constraint vertices* a,b,c, as well as y, are already in T. The cost of x is infinity if no valid edges exist for x.

In each iteration while Q is not empty, we perform the following operations. (1) Extract the current minimum-cost vertex v from Q, and mark v as included into T. (2) Update the key values of the vertices in Q that are "influenced" by v. Namely, we need to find the edges of G that become *valid* by the addition of v into T, and perform the necessary *decrease-key* operations on the corresponding vertices in Q. The candidates for such newly valid edges can be classified into two types: the edges of G incident on v, and the edges of G with v a constraint vertex. In either case, we can find such candidate edges via bidirectional links from v, and use the bidirectional links from the candidate edges to the corresponding vertices to check their marking status. Note that in the entire process each edge of G is examined at most five times, so that the overall complexity is the same as the original Prim's algorithm.

In case the extracted minimum-cost vertex v has infinite cost (meaning that v cannot be added to T via a valid edge), we start a new tree by the same process (such case occurred very rarely in our experiments). In this way, we can always span all the vertices.

3.3 Building a Pseudo-CMST and Final Encoding

Now we describe the last step of our encoding algorithm. From Section 3.2, we see that traversing the constructed CMST T by the order we grow T is a *feasible* flipping traversal. However, such order grows the boundary of the subvolume spanned by the current T *arbitrarily* according to which boundary face is the least expensive to flip across, and thus encoding such traversal requires us to specify which of the candidate boundary faces to flip across. As the number of such boundary faces is large, this is too expensive to encode. This is why we seek an alternative traversal and build a pseudo-CMST to get a better compression efficiency.

Data	# tetra.	# vert.	MST	CMST	Diff.
Spx	12936	20108	8.94	10.02	12.08%
Blunt	187395	40960	8.06	8.66	7.44%
Comb	215040	47025	6.16	6.24	1.30%
Post	513375	109744	7.82	8.13	3.96%
Delta	1005675	211680	6.64	7.00	5.42%
Cyl1	615195	131072	5.81	6.27	7.92%
Cyl10	615195	131072	13.09	13.46	2.83%

Table 1: Comparison of entropy of prediction errors (including x-, y-, z- and scalar values, in bits per vertex, b/v) with constrained and unconstrained MSTs.

The main idea of the algorithm is that each tetrahedron has at most four faces to flip across, which is cheap to encode, and each potential flipping corresponds to an edge $e \in G$. The constraint vertices of e are among the vertices of the current tetrahedron and are already visited, and thus the causality constraint is satisfied. If the face-neighboring tetrahedron t corresponding to e has been visited before then we ignore t. Otherwise, t is unvisited and let v be the new vertex of t to be predicted by e. If v has not been visited and e belongs to our constructed CMST T, then we include e into the pseudo-CMST, visit the new tetrahedron t, predict v via e, code the prediction error, and recursively proceed from t. If v has not been visited but e is *not* an edge of T, then we ignore t and e—v will not be predicted via e (and tetrahedron t and vertex v will be traversed later from some other paths). In the remaining case, where v has been visited before, we *always* include e into the pseudo-CMST with *no prediction cost*, visit the new tetrahedron t and proceed recursively from t. Such no-cost edge e is called a *pseudo-edge* (and hence the pseudo-CMST).

In general, there are three faces to expand from the current tetrahedron (four faces for a starting tetrahedron). Our algorithm recursively expands in one direction, and the next, in a depth-first manner. For the purpose of connectivity compression, we also need to collect those left-over tetrahedra not visited by the above process. Such tetrahedron-collection operations are interleaving with the above process so that we avoid the cost of specifying where to "insert" the left-over tetrahedra. When a new tetrahedron t (with new vertex v) is visited, we try to recursively expand from the face-neighboring tetrahedra of t, one by one, by applying the above process. When this is complete, we look at all the tetrahedra sharing v that are only edge-neighbors of t or only vertex-neighbors of t, and collect those that are left-over but have all four vertices already visited. Note that we do *not* recursively expand from such left-over tetrahedra. When the algorithm can no longer proceed and there are still unvisited vertices, we start a new tree by the same procedure. In this way, all the vertices and tetrahedra are visited.

4. EXPERIMENTAL RESULTS

We have implemented our new algorithm, and experimented on the datasets listed in Table 1. These are all well-known tetrahedral volume datasets, where Cyl10 is time-varying with 10 time steps, and Cyl1 is the same dataset with one time step. In Spx, only 2896 vertices are referenced by the tetrahedra and thus we removed the remaining vertices in our

Data	xyz + scalar values			xyz only		
	Flip	PseT	Gain	Flip	PseT	Gain
Spx	13.30	10.02	24.7%	9.88	7.26	26.5%
Blunt	11.17	8.70	22.1%	8.63	6.72	22.1%
Comb	8.04	6.24	22.4%	6.42	4.73	26.3%
Post	10.85	8.15	24.9%	8.65	6.40	26.0%
Delta	9.62	7.01	27.1%	8.11	5.78	28.7%
Cyl1	8.97	6.27	30.1%	7.49	5.23	30.2%
Cyl10	18.19	13.46	26.0%	–	–	–
Ave			25.3%			26.6%

Table 2: Comparison of entropy of prediction errors (in b/v) with different flipping approaches. 'PseT' is our pseudo-CMST, and 'Flip' is a state-of-the-art flipping method.

experiments as they cannot be predicted by the flipping operations. The vertex coordinates and scalar values are each quantized to an 8-bit integer before compression.

The first set of experiments was to evaluate the efficiency of our heuristic for computing an approximate *constrained* MST (CMST) by comparing it with an *unconstrained* MST, which is a lower bound (albeit unachievable). In Table 1 we see that the first order entropy of the prediction errors we get is almost always within 8% of that achieved by an unconstrained MST. On an average the difference is 5.85%. This shows that our resulting CMST is very close to optimal.

For the purpose of comparisons, we also implemented a state-of-the-art flipping approach, which traverses the tetrahedra using the connectivity coder of [25] and predicts the new vertices via the same flipping operation. In Table 2, we compare the compression performance of our prediction technique with this flipping approach. We see that when including the scalar values, our technique can be up to 30.1% more efficient (and 25.3% more efficient on an average), while the improvements are up to 30.2% (and 26.6% on an average) when not including the scalar values, showing a significant advantage of our algorithm.

REFERENCES

[1] P. Alliez and M. Desbrun. Valence-driven connectivity encoding for 3D meshes. *Computer Graphics Forum*, 20(3):480–489, 2001. Special Issue for Eurographics '01.

[2] D. Chen, Y.-J. Chiang, N. Memon, and X. Wu. Optimized prediction for geometry compression of triangle meshes. In *Proc. IEEE Data Compression Conference*, 2005 (to appear).

[3] D. Cohen-Or, R. Cohen, and R. Irony. Multi-way geometry encoding. TR-2002.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2nd edition, 2001.

[5] M. F. Deering. Geometry compression. In *Proc. ACM SIGGRAPH*, pages 13–20, 1995.

[6] O. Devillers and P.-M. Gandoin. Geometric compression for interactive transmission. In *Proc. Visualization '00*, pages 319–326, 2000.

[7] P.-M. Gandoin and O. Devillers. Progressive lossless compression of arbitrary simplicial complexes. *ACM Trans. Graphics*, 212(3):372–379, 2002. Special Issue for SIGGRAPH '02.

[8] S. Gumhold, S. Guthe, and W. Straser. Tetrahedral mesh compression with the cut-border machine. In *Proc. Visualization '99*, pages 51–58, 1999.

[9] S. Gumhold and W. Straser. Real time compression of triangle mesh connectivity. In *Proc. ACM SIGGRAPH*, pages 133–140, 1998.

[10] M. Isenburg. Compressing polygon mesh connectivity with degree duality prediction. In *Proc. Graphics Interface*, pages 161–170, 2002.

[11] M. Isenburg and P. Alliez. Compressing hexahedral volume meshes. In *Proc. Pacific Graphics*, 2002.

[12] M. Isenburg and P. Alliez. Compressing polygon mesh geometry with parallelogram prediction. In *Proc. Visualization*, pages 141–146, 2002.

[13] M. Isenburg and S. Gumhold. Out-of-core compression for gigantic polygon meshes. *ACM Trans. Graphics*, 22(3):935–942, 2003. Special Issue for SIGGRAPH '03.

[14] M. Isenburg and J. Snoeyink. Face fixer: Compressing polygon meshes with properties. In *Proc. ACM SIGGRAPH*, pages 263–270, 2000.

[15] Z. Karni and C. Gotsman. Spectral compression of mesh geometry. In *Proc. ACM SIGGRAPH*, pages 279–286, 2000.

[16] B. Kronrod and C. Gotsman. Optimized compression for triangle mesh geometry using prediction trees. In *Proc. Sympos. on 3D Data Processing, Visualization and Transmission*, pages 602–608, 2002.

[17] H. Lee, P. Alliez, and M. Desbrun. Angle-analyzer: A triangle-quad mesh codec. *Computer Graphics Forum*, 21(3):383–392, 2002. Special Issue for Eurographics '02.

[18] R. Pajarola, J. Rossignac, and A. Szymczak. Implant sprays: compression of progressive tetrahedral mesh connectivity. In *Proc. Visualization*, pages 299–306, 1999.

[19] J. Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Trans. Visualization Computer Graphics*, 5(1):47–61, 1999.

[20] J. Schneider and R. Westermann. Compression domain volume rendering. In *Proc. Visualization*, pages 293–300, 2003.

[21] MPEG-4 Standard. http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm.

[22] A. Szymczak and J. Rossignac. Grow & fold: compression of tetrahedral meshes. In *Proc. Solid Modeling and Applications*, pages 54–64, 1999.

[23] G. Taubin and J. Rossignac. Geometric compression through topological surgery. *ACM Trans. Graphics*, 17(2):84–115, 1998.

[24] C. Touma and C. Gotsman. Triangle mesh compression. In *Proc. Graphics Interface*, pages 26–34, 1998.

[25] C.-K. Yang, T. Mitra, and T.-C. Chiueh. On-the-fly rendering of losslessly compressed irregular volume data. In *Proc. Visualization '00*, pages 101–108, 2000.