Optimized Prediction for Geometry Compression of Triangle Meshes*

Dan Chen[†] Yi-Jen Chiang[‡] Nasir Memon[§] Xiaolin Wu[¶]

Abstract

In this paper we propose a novel geometry compression technique for 3D triangle meshes. We focus on a commonly used technique for predicting vertex positions via a flipping operation using the parallelogram rule. We show that the efficiency of the flipping operation is dependent on the order in which triangles are traversed and vertices are predicted accordingly. We formulate the problem of optimally (traversing triangles 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 17.4% on average as compared to the *unconstrained* minimum spanning tree which is an unachievable lower bound. We also show significant improvements over previous techniques in the literature that strive to find good traversals that also attempt to minimize prediction errors obtained by a sequence of flipping operations, albeit using a different approach.

1. Introduction

Recent advances in three-dimensional acquisition, simulation, and design technologies have led to the generation of large polygonal meshes in 3D graphics. The emerging demand for efficiently storing, transmitting, and visualizing such data in networked environments has motivated graphics compression for 3D polygonal models to become a focus of increasing research in the past several years. The most common polygonal models are triangle meshes, consisting of the following two components: *geometry*—the 3D coordinates of the mesh vertices, and *connectivity*—the incidence information specifying the edges and triangle faces connecting the mesh vertices.

Although there has been a significant amount of research done on graphics compression, most previous techniques focus on compressing the *connectivity*, rather than on the *geometry* information. As a result, while connectivity compression already achieves an impressive compression rate of 1.5–4 bits per vertex on average for triangle meshes [19, 17, 1, 20], progress

*The authors are with the Computer and Information Science Department at Polytechnic University, Brooklyn, NY 11201. Contact author is Yi-Jen Chiang.

†`dchen@cis.poly.edu`. Research supported by NSF Grant ACI-0118915.

‡`yjc@poly.edu`. Research supported in part by NSF Grant ACI-0118915, NSF CAREER Grant CCR-0093373, and NSF ITR Grant CCR-0081964.

§`memon@poly.edu`. Research supported in part by NSF Grant ACI-0118915 and NSF Grant CCR-0208678.

¶`xwu@poly.edu`. Research supported in part by NSF Grant CCR-0208678.

made in compressing geometry information has not been equally impressive. Typically, a floating-point vertex coordinate is quantized to a 10-bit or 12-bit integer (and hence 30 or 36 bits per vertex for the raw information), and the published techniques that deal with geometry compression (e.g., [3, 13, 19, 20]) generate codes whose size is 40-50% of such raw information. Clearly, geometry compression is the major 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 3D triangle meshes. Our approach is based on the *flipping* algorithm first introduced by Touma and Gotsman [20], which traverses the triangles and predicts the position of the next vertex via a *flipping* operation using the *parallelogram rule*. This flipping algorithm [20] 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 [18]. However, like almost all other previous approaches, the original flipping algorithm [20] focuses on connectivity compression and traverses triangles in an order that ignores the geometry of the model, resulting in a geometry compression ratio that is far below optimal as we show in this paper.

The basic idea behind our approach is the observation that for each vertex, there are many different ways to predict its geometry position via the parallelogram-rule flipping operation. We formulate the problem of optimally (traversing triangles and) predicting the vertices via flippings as an optimization problem of a combinatorial nature, namely, the *minimum-spanning-tree* (*MST*) problem on a graph whose nodes are the mesh vertices and whose edges are all possible flippings with edge cost being the prediction error. However, not all flipping sequences are realizable due to causality constraints. That is, we cannot predict a vertex by a flipping operation unless it is the only unvisited vertex of the parallelogram that defines the underlying flipping operation. Hence we introduce the notion of a *constrained* MST (*CMST*) which admits a traversal that does not violate causality constraints. We develop a heuristic algorithm to construct an (approximate) CMST, whose cost in practice (as shown in our experiments) is only 17.4% more on average than the cost of the *unconstrained* MST, which certainly is a severe (unachievable) lower bound on the optimal cost of CMST. Moreover, the flipping operation imposes the causality constraints on the order to *traverse* the constructed CMST; traversing it according to the order in which the tree is constructed is feasible but very expensive to encode. We further develop an encoding algorithm that traverses the constructed tree in another pass, which reduces the encoding cost significantly. The "left-over" triangles that are not yet visited but whose vertices are already spanned are "collected" (for the purpose of connectivity compression) in this same tree-traversal pass. In this way, our compression technique produces a geometry code that is close to optimal (in the sense of what can be achieved by using the flipping operation for prediction), at a slight expense of the cost of the connectivity code. Also, while the encoding step needs more than one pass, decoding needs only one pass and takes linear time.

The rest of the paper is organized as follows. In Section 2 we review the previous work. In Section 3 we describe our new approach, including the problem formulation and the proposed algorithm. Finally, we present our experimental results, which show the significant advantages of our compression method over the previous best cover-tree approach [15] (up to 33.2% more compact) and over the widely-used benchmark arbitrary-flipping approach [20] (up to 55.4% more compact), in Section 4.

2. Previous Related Work

There has been a significant amount of work on compressing polygonal meshes [3, 19, 7, 17, 12, 1, 9, 13, 16, 20]. Much of this work has mainly focussed on compressing *connectivity* information. As mentioned before, these techniques achieve an impressive compression performance of 1.5–4 bits per vertex on average for triangle meshes [19, 17, 1, 20].

There are relatively few results that focus on compressing geometry information. Lee *et. al.* [16] proposed the *angle analyzer* approach for traversing and encoding polygonal meshes consisting of triangles and quads. Devillers and Gandoin [4, 5] developed compression techniques that are driven by the geometry information, for both triangle meshes and tetrahedral meshes. The techniques are for *multi-resolution* compression, rather than for *single-resolution* compression as we consider in this paper. (We refer to [5, 8, 14] and references therein for other multi-resolution compression techniques.)

The most popular technique for compressing geometry information of polygonal meshes is the *flipping* algorithm based on the *parallelogram rule* first introduced by Touma and Gotsman [20] as mentioned before. Isenburg and Alliez [10] extended the parallelogram rule so that it works well for polygonal surfaces beyond triangle meshes. Isenburg and Gumhold [11] applied the parallelogram rule in their out-of-core compression algorithm for polygonal meshes larger than can fit in main memory.

The only previous geometry compression approach that tries to optimize the flipping prediction cost is the work by Kronrod and Gotsman [15] via the *prediction tree*. They formulate the problem differently, however, as finding an optimal *cover tree*, which intuitively is trying to (partially) span the mesh-triangle nodes in the dual graph of the triangle mesh until all mesh vertices are covered, with minimum total dual-edge weight. The optimal cover-tree problem is a special case of the well-known NP-hard Minimal Steiner Tree problem for weighted directed graphs [6, 15], and they solve it with a factor-2 approximation heuristic algorithm for undirected Minimal Steiner Tree problem [15]. There are two major differences in this cover-tree formulation from ours: first, the mesh vertices may be covered more than once, as opposed to being spanned exactly once in our case; secondly, a triangle can only be visited (and used for prediction) from an edge-adjacent neighboring triangle, and can never be visited from a vertex-adjacent triangle as is possible in our case. Consequently, our compression results can be up to 33.2% better as shown in the experiments.

3. Our Approach

In this section we present the technical details of our algorithm. We first formulate the problem and give an overview of our method, and then describe its detailed steps.

3.1 Problem Formulation and Algorithm Overview

As mentioned before, our algorithm is based on the flipping algorithm [20], which traverses the triangles of the mesh and predicts the geometric position of the next vertex via the *flipping* operation using the *parallelogram rule*, as follows (see Fig. 1(a)). Suppose we already traversed $\triangle abx$ (the triangle represented by the vertices a, b and x); when we continue to traverse its neighbor $\triangle aby$, we predict the position of the new vertex y as at y', where

(a, x, b, y') form a parallelogram (and hence (x, y') goes through the center c of (a, b), with (x, c) and (c, y') having the same length). We say that we predict y from x via (a, b), and the vector (y', y) defines the prediction error/cost. Similarly, we can predict x from y via (a, b). Notice that the prediction-error vectors (x', x) and (y', y) are the same (since $\triangle xx'c$ and $\triangle y'yc$ are congruent).

The basic idea behind our approach is the observation that for each vertex, there are many different ways to predict its position, with different prediction costs. For example on Fig. 1(b) we see that vertex v can be predicted via a flipping operation via any of the vertices v_1, v_2, v_3, v_4, v_5. If the goal is compression then we need to predict each vertex *once*, and minimize the total prediction error for the entire mesh. This problem can be formulated as a combinatorial optimization problem, namely, finding a *minimum spanning tree* (*MST*) on the graph $G = (V, E)$, where the nodes in V are the mesh vertices, and the edges in E connect two nodes (mesh vertices) x and y if and only if x and y can be predicted from each other (see Fig. 1(a)). Since the prediction errors (x', x) and (y', y) are the same, all edges $(x, y) \in E$ are *undirected* with edge costs the prediction error (x', x) (hence G is an *undirected*, weighted graph).

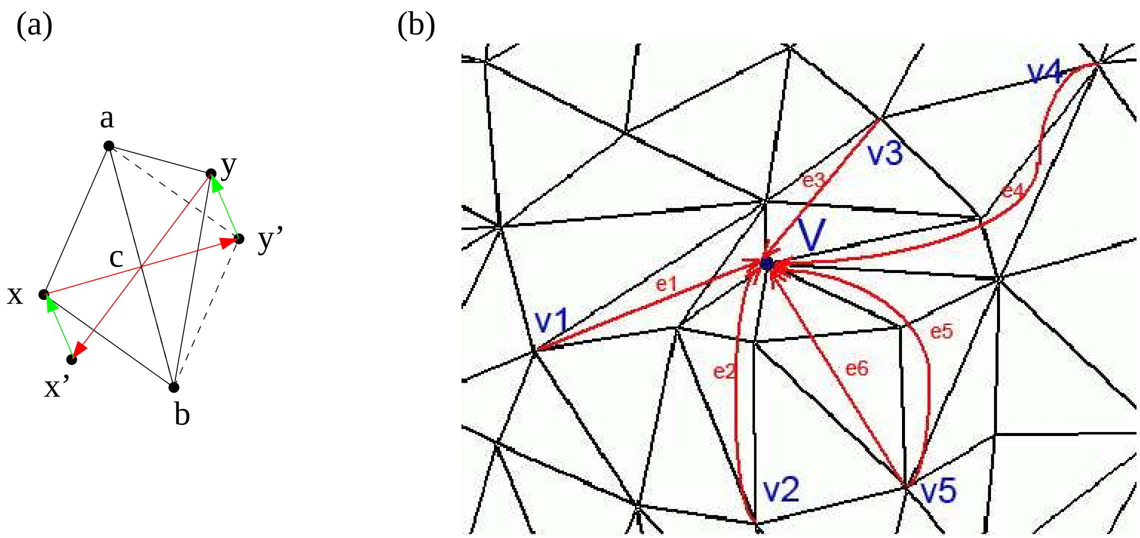

Figure 1: The flipping prediction using the parallelogram rule. (a) We predict y from x via (a, b) as at y', with prediction error/cost (y', y). (b) The vertex v, with degree 6, can be predicted by 6 different ways $e_1, ..., e_6$, with different prediction costs.

The above MST problem on the graph G is not yet the correct formulation of the problem, however. Consider the MST in Fig. 2, where we start with $\triangle v_1 v' v''$ and span vertices $v_1, ..., v_6$ with the edges $(v_1, v_2), (v_2, v_3), ..., (v_5, v_6) \in E$. This MST is *not feasible* for prediction, since in order to predict v_3 from v_2, we need to know the position of v_a first, which is not yet visited. Similarly, v_a and v_b need to be visited before we can predict v_4 from v_3, and so on. In other words, we cannot predict a vertex by a flipping operation unless it is the only unvisited vertex of the parallelogram that defines the underlying flipping operation. 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. We show in Fig. 3 an example of CMST and its realizable prediction traversal.

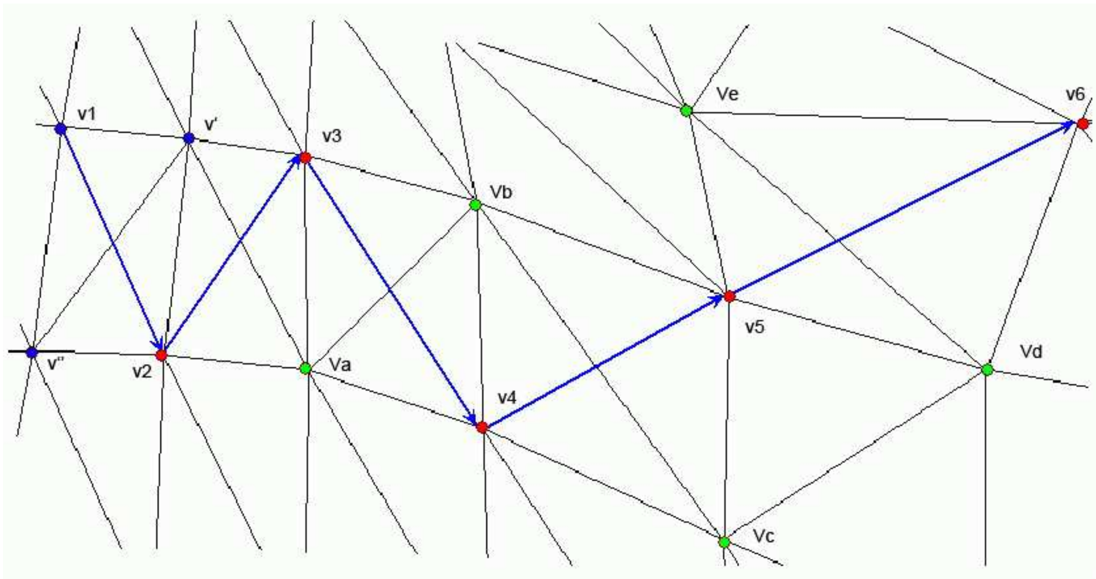

Figure 2: The constraint of the flipping operation makes the MST shown (the path of arrows connecting $v_1, v_2, ..., v_6$) not feasible to use for prediction—we need to visit $v_a, ..., v_e$ first before predicting $v_3, ..., v_6$.

To solve the CMST problem on G, we develop a heuristic algorithm to compute an (approximate) CMST. Our next task is to use the constructed tree to traverse triangles and predict vertices accordingly, in an order that satisfies the causality constraint of the flipping prediction, to perform the geometry encoding. It turns out that traversing the tree in the order in which it is constructed is a realizable order but is very expensive to encode. We develop an encoding algorithm that traverses the constructed tree in another pass, and build a *pseudo-CMST*, which in fact is a forest where each tree can be efficiently encoded and permits a feasible flipping traversal. The pseudo-CMST may have a slightly higher prediction cost than the constructed CMST because of the starting cost of each tree in the forest. However, the overall encoding cost of pseudo-CMST is significantly reduced from direct encoding of the constructed CMST. Finally, since in the CMST-construction step we stop traversing triangles as soon as all mesh vertices are spanned, there are "left-over" triangles not yet visited. For the purpose of connectivity compression we need to collect such triangles as well. Our algorithm performs such triangle-collection step in the same tree-traversal pass, and finishes both geometry and connectivity encoding simultaneously, in an interleaving fashion.

In summary, our algorithm performs the following steps.

1. Form the graph G.

2. Construct an approximate CMST on G by a heuristic algorithm.

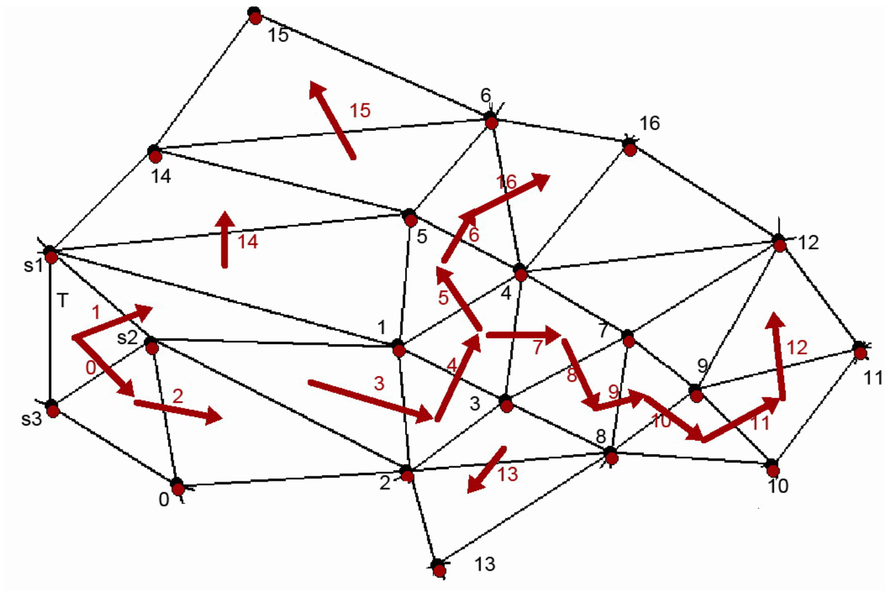

Figure 3: An example of a constrained MST (CMST). Each arrow goes between two edge-adjacent triangles involved in a flipping prediction. The numbers (labeling the arrows and the corresponding predicted vertices) indicate the order the vertices are predicted/visited. The initial triangle is $T = \triangle s_1 s_2 s_3$.

3. Traverse CMST in another pass, build a pseudo-CMST and collect the left-over triangles in an interleaving fashion, and finish both geometry and connectivity coding.

In the following, we describe Steps 2 and 3 in more detail.

3.2 Heuristic Algorithm for Constrained Minimum Spanning Tree

Now we present our heuristic algorithm to find an approximate constrained minimum spanning tree. Referring to Fig. 1(a), the flipping traversal constraint means that in order to predict from x to y or from y to x, the two mesh vertices a and b must have been visited before. In terms of minimum spanning tree construction, this means that in order to include the edge $(x, y) \in G$ into the spanning tree T, the two nodes a and b of G must have been in T already. We establish a *bidirectional* correspondence between such *constraint vertices* a, b and edge (x, y), for each edge $(x, y) \in G$. Note that (x, y) has a bidirectional link to each of a and b, so that a vertex a also knows the edges in G constrained by a.

Our heuristic algorithm is a *modified* Prim's algorithm [2]. Initially we include the three vertices of a triangle into T, which enables initial predictions. We may first find the edge $e \in G$ with the smallest prediction cost, and choose one of the two triangles that can enable this flipping. Throughout the process, we use a priority queue Q to maintain 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, as well as y, are already in T. The cost of x is infinity if no such 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. Observe that in the entire process each edge of G is examined no more than four 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. Also, as will be seen in Section 4, the overall cost of our resulting spanning tree is close to the cost of an *unconstrained* MST, which is an unachievable lower bound.

3.3 Pseudo-CMST Calculation and Final Encoding

Now we describe the last step of our encoding algorithm. It is easy to see from Section 3.2 that traversing our constructed CMST T according to the order we grow T is a *feasible* flipping traversal. However, such order grows the boundary of the patch spanned by the current T *arbitrarily* according to which boundary portion is the least expensive to grow (via the flipping prediction), and thus to encode such traversal requires us to specify from which of the candidate boundary edges to grow. As the number of such boundary edges is large, this is too expensive to encode. This is why we need to traverse the constructed CMST T in another pass and build a pseudo-CMST for more efficient encoding.

The basic idea of our algorithm is that each triangle has at most three edges to flip across, and each potential flipping corresponds to an edge $e \in G$. Note that the constraint vertices of e are the vertices of the current triangle and thus are already visited. If the edge-neighboring triangle 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 before and e belongs to our constructed CMST T, then we include e into the pseudo-CMST, visit the new triangle t, predict v via e, code the prediction error, and recursively proceed from t. If v has not been visited before but e is *not* an edge of T, then we ignore t and e—v will not be predicted via e (and triangle 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 triangle 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 two directions to expand from the current triangle (three directions for a starting triangle). 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 triangles not visited by the above process. Such triangle-collection operations are interleaving with the above process so that we avoid the cost of specifying where to "insert" the left-over triangles. When a new triangle t (with new vertex v) is visited, we try

Model	# of Vertices	# of Triangles	MST (b/v)	CMST (b/v)	Difference (%)
apple	867	1704	13.29	14.04	6
banana	272	512	13.59	15.15	11
bishop	250	496	15.34	16.5	8
bolt	328	652	11.12	14.29	29
bunny	1494	2915	19.68	20.41	4
cow	2894	5504	13.85	15.01	8
dolphins	855	1689	16.05	16.95	6
electro	6546	13080	4.53	7.42	64
funnel	4140	8279	6.35	9.09	43
rose	2346	3576	15.03	16.8	12
snail	760	720	18.15	19.71	9
triceratops	2832	5160	10.21	12.21	20

Table 1: Comparison of entropy of prediction errors (in bits per vertex, b/v) with constrained and unconstrained MSTs.

to recursively expand from the edge-neighboring triangles of t, one by one, by applying the above process. When this is complete, we look at all the triangles sharing v that are only vertex-neighbors of t, and collect those that are left-over but have all three vertices already visited. Note that we do *not* recursively expand from such left-over triangles. 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 triangles are visited in the same pass.

4. Experimental Results

In this section we present experimental results comparing our techniques with previous state-of-the-art approaches. We performed our experiments using 12 different 3D models that are commonly used in the literature[1]. The models range from small to moderately large and from smooth to ones containing significantly many sharp corners. The vertex coordinates were quantized to 12-bit integers as is the common practice in the graphics literature.

The first experiment we performed was with the goal of comparing our heuristic for computing a *constrained* MST (CMST) with an *unconstrained* MST. The rationale here is to get a lower bound (albeit unachievable) for the CMST and compare the performance of our heuristic against this lower bound. It can be seen from Table 1 that a large number of times the first order entropy of prediction errors we get is often within 10% of that achieved by an unconstrained MST. On average the difference is 17.4%. It is interesting to observe that for three models which are highly irregular (electro, funnel and bolt) the difference is the largest. This we conjecture is due to the bound being loose for such models and the lack of a feasible traversal of the unconstrained MST. Our ongoing work is to address this observation.

[1]We would like to thank Craig Gotsman for providing the models in the list that were used for reporting results in [15].

Model Name	CMST (b/v)	TG-flip (b/v)	KG-PT (b/v)	Pseudo CMST (b/v)	Gain over KG-PT (%)	Gain over TG-flip (%)
apple	13.29	21.00		14.11		32.81
banana	13.59	20.19		15.12		25.11
bishop	15.34	22.45		16.78		25.26
bolt	14.2	21.6	18.9	14.7	22.33	32.04
bunny	20.4	24.2		20.41		15.80
cow	15.0	19.8	18.8	15.1	19.95	23.99
dolphins	16.05	23.98		17.00		29.11
electro	7.4	15.7	7.7	8.2	-6.23	47.90
funnel	9.1	21.3	14.2	9.5	33.17	55.45
rose	15.03	27.56		16.82		38.97
snail	18.15	26.69		19.80		25.81
triceratops	12.2	19.4	17.3	12.5	28.03	35.82

Table 2: Comparison of entropy of prediction errors with different flipping approaches.

Next we compared the performance of our prediction technique to those reported in the literature. Column 5 (labeled Pseudo CMST) in Table 2 lists our results on the same models we used above. The results in column 3 (labeled TG-flip) are for arbitrary flipping we obtained by running a copy of the executable code provided by the authors of [20] and available at http://www.cs.technion.ac.il/c̃s234325/Homepage/Software-Sources/Gotsman/Vrml/vrml/. The results given in column 4 (labeled KG-PT) for the prediction-tree flipping technique by Kronrod and Gotsman [15] are listed from their paper. It can be seen that the approach proposed in this paper gives large improvements over the two state-of-the-art techniques. The improvement over arbitrary flipping is up to 55.45% (and more than 32% on average), and over the more sophisticated prediction-tree flipping is up to 33.17% (and more than 18% on average). Finally it can also be seen that the encoding cost incurred by pseudo-CMST is not much more than the original prediction cost of CMST given in column 2.

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

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

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

[5] 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.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* pages 208–209. W. H. Freeman, New York, NY, 1979.

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

[8] H. Hoppe. Efficient implementation of progressive meshes. *Computers & Graphics*, 22(1):27–36, 1998.

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

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

[11] 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.

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

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

[14] A. Khodakovsky, P. Schröder, and W. Sweldens. Progressive geometry compression. In *Proc. ACM SIGGRAPH*, pages 271–278, 2000.

[15] 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.

[16] 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.

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

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

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

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