Lossless Compression of Point-Based 3D Models

Dan Chen* Polytechnic University, NY
Yi-Jen Chiang† Polytechnic University, NY
Nasir Memon‡ Polytechnic University, NY

1. Introduction

With the increasing sizes and complexity of the 3D models, it is an emerging demand to develop graphics compression techniques in order to efficiently store, transmit, and visualize such data in networked environments. Although there has been a very rich literature in graphics compression, most techniques developed so far have focused on compressing *polygonal meshes*, where even the state-of-the-art *geometry* coders for compressing the vertex information [3, 6, 7, 10] require the use of the connectivity information and thus are not applicable to point-set compression. Among the relatively few results on point-set compression, the technique of [11] mainly focuses on *multiresolution* (rather than *single-resolution*) compression, and those of [4, 8] require *resampling* which alters the datasets and may not be allowed in some applications. Clearly, point-set compression is still in its early stage.

In this paper, we present a single-resolution technique for lossless compression of point-based 3D models. An initial quantization step is *not needed* and we can achieve a truly lossless compression. The scheme can compress geometry information as well as attributes associated with the points. We employ a three-stage pipeline that uses several ideas including k-d-tree-like partitioning, minimum-spanning-tree modeling, and a two-layer modified Huffman coding technique based on an optimal alphabet partitioning approach using a greedy heuristic. We show that the proposed technique achieves excellent lossless compression results.

2. Our Approach

A widely used technique for point-set compression is differential coding, i.e., the differences between adjacent vertices are encoded. Our main idea is to *re-order* the vertices so as to minimize the total bits needed to encode the overall differences. In addition, we can compute a difference with respect to not just an immediate predecessor, but to *any* previously visited vertex. This opens up new avenues for optimization of vertex re-ordering. Moreover, we need an efficient coding technique to encode the differences. We discuss these in more details.

2.1. Vertex Re-ordering

We observe that representing a set of points using a minimal set of pairwise differences essentially involves computing a spanning tree of the complete graph defined by the points. Specifically, we form a complete weighted graph G where the nodes are the vertices $v_1, \cdots, v_n$ of the point cloud and the weight on each edge is the coordinate-wise 3-tuple of differences. That is, if $v_i = (x_i, y_i, z_i)$ and $v_j = (x_j, y_j, z_j)$ then the edge (v_i, v_j) has weight $(x_i - x_j, y_i - y_j, z_i - z_j)$. The tuples of v_i and v_j are extended similarly when attribute values are included, with the weight defined analogously.

Definition 1 *Given a spanning tree T of a weighted graph G, let t_i be the number of times a distinct edge weight w_i appears in the edge weights of T, and $\alpha_i = t_i / \sum_i t_i$. Then the* zero-order entropy *of the weights on the edges of T is defined by $H_0(T, G) = -\sum_i \alpha_i \log \alpha_i$.*

Over all possible spanning trees of G, there exists a tree T that minimizes the entropy $H_0(T, G)$:

Definition 2 *Given a weighted graph G, we define a minimum entropy prediction tree (ME tree) to be a spanning tree T such that $H_0(T, G) \leq H_0(T', G)$ for all spanning trees T' of G.*

Now the problem is: how do we compute an ME tree of G? We can prove that this problem is *NP-Hard*, by reducing the problem of finding an exact three cover by 3-sets (X3C problem, known to be NP-Complete [5]) to our ME-tree problem. We do not provide a proof here for a lack of space, but will do so in the full paper.

Since the ME-tree problem is NP-Hard, we next look at special conditions where it may admit a tractable solution. To do this we first define the *Minimum Absolute Weight* (MAW) tree to be a spanning tree that minimizes the sum of the absolute value of the weights on its edges. If we assume that smaller weights occur with higher frequency then we can show that the ME tree is indeed an MAW tree. Again, we skip the proof and defer that to the full paper. Since an MAW tree can

*dchen@cis.poly.edu. Supported by NSF CCF-0118915.
†yjc@poly.edu. Supported in part by NSF CCF-0118915, NFS CAREER CCF-0093373, and NFS ITR CCF-0081964.
‡memon@poly.edu. Supported in part by NSF CCF-0118915.

be computed efficiently using the well-known Kruskal's or Prim's minimum-spanning-tree (MST) algorithm, we now have an efficient way to compute a desired spanning tree T. Note, however, that T will not be known to the decoder and must be efficiently encoded and sent as side information. We describe how to do this next.

2.2. Encoding the Minimum Spanning Tree

Now we describe our technique for encoding the computed minimum spanning tree T. We remark that the previous *run-length coding* approach in [9] may not work well since there might be many runs in our more general setting, and thus a new tree-encoding method is needed.

Our technique is simple and general; it is based on a *breadth-first-search (BFS)* traversal of T. We first pick an arbitrary node r of T as the root, and then start a BFS traversal at r. At the beginning, we put down the *original* data of r, and the number n_r of child nodes of r, followed by n_r *differences*, each a child c_i of r represented by the *difference* $c_i - r$. We then traverse all children of r one by one as enumerated, and then all grand-children, and so on, using the same process. Notice that if node v is a leaf, we still put down $n_v = 0$, with *no* difference following n_v. All the differences will be further encoded by our additional coding approach described in Section 2.3. For the numbers n_v, we simply use a *fixed-length* coding. This means that in addition to the cost of our differential coding for the data, the overhead of encoding the tree is $\log_2(\max_v n_v)$ bits per vertex, where n_v is the number of children of node v in T. In all our experiments the maximum number of children is 4, and hence this overhead is only 2 bits per vertex. The decompression process follows the compression scheme; we additionally maintain a pointer to the next vertex v whose children are yet to be generated, so that the correct offset can be added to the differences to recover the children of v. Clearly, each encoding/decoding of the tree takes optimal linear time.

2.3. The Overall Algorithm

Our overall algorithm consists of the following steps.

Step 1. Integer Mapping We map the coordinates and attribute of the vertices to integers by an invertible operation. Specifically, we can take a 32-bit floating-point representation and simply treat it as a 32-bit integer, which clearly can be mapped back to the original floating point.

Step 2. Vertex Partitioning We partition the vertices into k^3 clusters for a given parameter k using a *k-d-tree-like* scheme as follows. We sort the vertices by x-values and split them equally into k groups; for each group we repeat the process by y-values, and finally we repeat the process again by z-values. The resulting k^3 groups are the k^3 clusters. The purpose of this step is to speed up Step 3 (and keep compression efficiency) by a suitable k.

Step 3. Vertex Re-ordering For each cluster, we re-order the vertices using the MST approach in Sections 2.1–2.2 to minimize the entropy of the vertex differences.

Step 4. Entropy Coding Typically the vertex differences have a very high dynamic range (i.e., they have a large alphabet size), and cannot be efficiently encoded in a straightforward way. In our recent work [2] we resolved this issue by developing a two-layer modified Huffman coding technique that minimizes the total cost of encoding the data as well as the data-specific Huffman table using an optimal alphabet partitioning method based on a greedy heuristic. Here, we pool the differences across all coordinates and all clusters, and construct a *single* modified Huffman code using the greedy method of [2]. This results in a *single* Huffman table for each dataset and hence the Huffman-table cost is further reduced.

3. Experimental Results

In our experiments we used point-cloud datasets obtained by taking triangle meshes and removing their connectivity information. This gave us an additional freedom to compare our algorithm with state-of-the-art geometry compression technique [10] for triangle meshes (which utilizes the connectivity information to help obtaining efficient compression) and use this as a bound on the performance of our technique.

We used a total of thirteen models, available at `http://cis.poly.edu/~dchen/research.html`, as shown in Table 1. Each vertex coordinate is given as a 32-bit float (thus 96 bits per vertex). In all our experiments we partitioned the vertices into k^3 clusters for some integer k so that there were 100-200 vertices per cluster. This gave a fast and yet competitive compression among choices of k.

Lossless Compression Results obtained with our MST-based lossless compression are shown in Table 1. For the sake of comparison we also present results obtained with Gzip, a commonly used lossless compression technique, and another method based on TSP re-ordering which was originally proposed for tetrahedral meshes in our previous work [1] but can be easily adapted for point clouds. On examining the results we see that our MST approach gave the best compression performance. In particular it gave more than 17.8% improvement over Gzip on an average. We note that the entropy coding technique used for the MST and TSP methods was the alphabet partitioning technique described in Section 2.3, where only one Huffman table was constructed for each dataset.

Comparison with Flipping To gauge the efficacy of our compression algorithm, we compare our method

Dataset	MST bpv	TSP bpv	Gzip bpv	MST Time	TSP Time
armadilo	42.93	45.29	55.24	265	517
blade	30.97	31.24	41.88	3409	4245
bunny	70.71	70.46	87.64	53	71
dragon	60.26	63.09	66.04	73	170
drill	40.74	44.65	50.26	5	11
hand	39.80	40.00	45.43	822	1082
budda	45.53	46.87	57.09	1381	1558
horse	63.00	65.78	81.05	62	74
phone	65.91	65.89	86.31	135	155
rabbit	55.63	59.06	63.85	132	157
sdriver	59.11	63.69	70.25	56	66
teeth	54.15	56.89	60.93	185	220
venus	53.23	57.26	64.41	252	321

Table 1. Results for lossless compression (with *no quantization*) in bits per vertex (bpv) for three techniques including all overheads. Compression times are in seconds.

Name	*32bit quant*		*24bit quant*		*16bit quant*	
	MST	Flip	MST	Flip	MST	Flip
armadilo	44.7	67.2	38.4	45.3	22.4	24.4
bunny	75.2	74.6	54.9	51.4	27.6	20.7
dragon	60.0	71.6	55.1	57.1	33.8	28.6
drill	38.7	59.5	35.7	40.6	17.8	17.5
hand	38.9	73.7	35.1	50.6	21.9	24.7
budda	43.5	69.7	38.7	52.8	23.1	24.6
horse	60.6	71.9	55.7	54.3	33.8	26.0
phone	68.9	72.9	60.1	53.3	32.7	22.7
rabbit	57.4	70.3	53.5	54.2	33.8	26.5
sdriver	54.5	69.4	51.9	55.0	30.8	26.7
teeth	50.9	70.3	47.6	53.6	31.7	26.2
venus	56.0	67.1	51.1	51.6	32.5	24.5

Table 2. Compression results in bits per vertex (bpv) after quantization of original data.

for point-cloud compression with the well-known and widely-cited flipping method [10] that utilizes the connectivity information (albeit such information is not available for point clouds). Since the flipping method reported in the literature performs quantization prior to compression, we quantized each coordinate to a 16-, 24-, and 32-bit integer and used them for comparison. We implemented the flipping prediction method based on [10] and encoded the prediction errors by a simple strategy of Huffman coding. The results are shown in Table 2. Our results show that as the quantization increased from 32 to 16 bits, the efficiency of the flipping method was gradually increasing. Compared to flipping, our MST method performed 22.82% better on an average when quantized to 32 bits. But when quantization increased to 24 bits, the improvement was down to only 7%. When we quantized to 16 bits, the MST method actually did worse by 16% on an average. This shows that when the quantization precision is decreased, flipping is able to predict more accurately. We conclude that flipping is more suitable for lossy compression (after quantization) whereas for lossless compression our technique is more suitable.

It is to our surprise that our method performed better than flipping for higher-precision quantization compression. One may say that this is due to the superior entropy coding technique used in our algorithm. However, when we compared just the entropy of the prediction errors (taken a byte at a time), our approach still gave better performance. This does not mean that connectivity information is of no use; it just means that we still need to develop better prediction techniques that are able to better use this connectivity information in lossless compression.

References

[1] D. Chen, Y.-J. Chiang, N. Memon, and X. Wu. Lossless geometry compression for steady-state and time-varying irregular grids. Submitted for publication, 2005.

[2] D. Chen, Y.-J. Chiang, N. Memon, and X. Wu. Optimal alphabet partitioning for semi-adaptive coding of sources with unknown sparse distributions. In *Proc. Data Compression*, pages 372–381, 2003.

[3] D. Chen, Y.-J. Chiang, N. Memon, and X. Wu. Optimized prediction for geometry compression of triangle meshes. In *Proc. Data Compression*, pages 83–92, 2005.

[4] S. Fleishman, D. Cohen-Or, M. Alexa, and C.T. Silva. Progressive point set surfaces. *ACM Trans. Computer Graphics*, 22(4):997–1011, 2003.

[5] 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.

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

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

[8] T. Ochotta and D. Saupe. Compression of point-based 3D models by shape-adaptive wavelet coding of multi-height fi elds. In *Proc. Point-Based Graphics*, 2004.

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

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

[11] M. Waschbusch, M. Gross, F. Eberhard, E. Lamboray, and S. Wurmlin. Progressive compression of point-sampled models. In *Proc. Point-Based Graphics*, pages 95–102, 2004.