COMPRESSING MATERIAL AND TEXTURE ATTRIBUTES FOR TRIANGULAR MESHES

Guoping Wang[1], *Yongzhen Wu*[1], *Guojing Hu*[1], *Jingliang Peng*[2,*], *Member, IEEE*

[1]Beijing Engineering Research Center of Virtual Simulation and Visualization (Peking University)
[2]School of Computer Science and Technology, Shandong University
Emails: wgp@pku.edu.cn, wuyz145@gmail.com, sulepluto@gmail.com, jpeng@sdu.edu.cn

ABSTRACT

A novel scheme for single-rate compression of material and texture attributes for triangular meshes is proposed in this work. For the material coding, it proposes a novel approach based on breadth-first surface traversal, exploiting the local coherence of the material attributes among neighboring facets; for the texture coordinate coding, it proposes a similar-triangle-based prediction method, exploiting the correlation between 3D geometry and its texture-space parameterization. As a result, the proposed algorithm achieves efficient coding of the material and the texture attributes, as experimentally demonstrated.

Index Terms— Compression, material, texture, mesh

1. INTRODUCTION

Three-dimensional (3D) models have been increasingly used in many applications including 3D filming, video gaming, scientific visualization and so forth. At the same time, complex 3D models are being increasingly produced, enabled by advances in 3D scanning and modeling technology. Therefore, effective compression of 3D models becomes a key problem which has attracted intensive research attention in the past couple of decades.

Most of the published 3D mesh compression algorithms focus on efficient encoding of the connectivity and the geometry data. Only a very small portion of them addresses the compression of material and texture attributes. However, 3D models with material and texture attributes are routinely used for realistic rendering effects. These attributes consume a significant bit budget, posing an urgent need for effective compression of them. While it is frequently assumed that material and texture attributes can be encoded similarly to the encoding of vertex positions, it is often not the case especially when material and texture attributes are associated with facets or corners but not vertices in a 3D mesh.

In this work, we propose a scheme that effectively compresses the material and the texture attributes contained in a 3D triangular mesh. Compared with peering works, the major novelties of this work include:

- We for the first time address the issue of material encoding, to the best of our knowledge.
- We propose a similar-triangle-based prediction method for texture coordinate encoding, which exploits the correlation between 3D geometry and its texture-space parameterization.

The rest of this paper is organized as follows. A review on related work is given in Section 2 followed by the description of the proposed scheme in Section 3. Experimental results are given in Section 4 and conclusion drawn in Section 5.

2. RELATED WORK

A huge amount of research has been conducted on 3D Mesh compression, comprehensive surveys of which can be found in references [1, 2, 3].

While the majority of the research has focused on compression of the geometry and the connectivity information, attributes like material and texture coordinates make important contributions to realistic rendering and frequently consume large storage spaces and network bandwidths. However, there have been very few works on attribute encoding, some of which include [4, 5, 6, 7, 8, 9, 10, 11]. References [11, 10] focus on compressing color attributes, which are often associated with vertices in a 3D mesh. Reference [4] encodes normal and color attributes, which are associated with vertices, in a way similar to the encoding of vertex positions. References [5, 6] differentiate the ways of binding attributes with a mesh, *i.e.*, attributes can be associated with a mesh on a basis of per vertex, per face or per corner. For the per corner association, they use extra bits to indicate discontinuous corner. They use simple linear combination prediction for attributes coding but report no experimental results. References [7, 8] pay more attention to efficiently compressing the property mappings, whose size is however just a small portion of the mesh representation. Reference [9] predicts texture coordinates using the parallelogram rule to reduce the data entropy.

* Corresponding author.

3. PROPOSED METHOD

Both the material and the texture attributes coders are based on a breadth-first traversal guided by the local coherency of materials or texture mappings over the surface. Flag bits are used to indicate the continuities and discontinuities of materials and texture mappings encountered during the traversal, which are entropy encoded. New attribute data need to be encoded only where the flag bits indicate discontinuities. Further, simple and effective prediction techniques are devised for texture coordinates coding, where the texture coordinates of a corner can be predicted from those of topologically close corners which have been encoded already, resulting in significantly reduced data entropies.

3.1. Material Coder

Materials define the light reflecting properties of a surface for the purposes of rendering. A 3D model may use many materials and each facet is associated with a material. In order to encode the model with materials, we encode the material index for each facet. The material data themselves are quantized and arithmetic encoded separately.

Facets of the same material tend to form a continuous region on the mesh surface, which we call a material region. The proposed material index coding scheme is driven by a breadth-first traversal on each material region. Initially, we pick the facet with the lexicographically smallest vertex index tuple as the seed, encode its material index, mark its material index as encoded, and put it into a global queue. Then, an iterative process is performed. In each iteration, if the queue is not empty, we take the facet, $f_i = \triangle(v_{i,1}, v_{i,2}, v_{i,3})$ $(v_{i,1} < v_{i,2} < v_{i,3})$, out from the front of the queue, and examine each neighboring facet,f_j, of f_i. If f_j's material index has not been encoded, we check if it equals to f_i's. If it does, we encode a bit '0' and append f_j to the end of the queue; otherwise we encode a bit '1' and f_j's material index but do not append f_j to the queue. Thereafter, we mark f_j's material index as encoded. When the queue becomes empty, we pick another seed facet, if still any, whose material index has not been encoded, encode its material index, mark its material index as encoded, enter it to the queue, and perform the iterative coding process as described above. The above-described process is repeated till all the facets' material indices have been encoded. Essentially, the proposed material coder exploits the coherency in material for facets in a material region to avoid explicit coding of all facets' material indices.

An exemplar mesh is shown in Fig. 1 which is covered by three material regions as colored differently. The initial seed facet is the one marked with '1' which is put in the queue initially. In the 1st iteration, the seed facet is removed from the queue and its neighboring facets marked with '2' are appended to the queue, and so forth. Note that in the 2nd iteration, the facets marked with '3' are appended to the queue while the two facets framed with dash lines are not since they belong to a different material region.

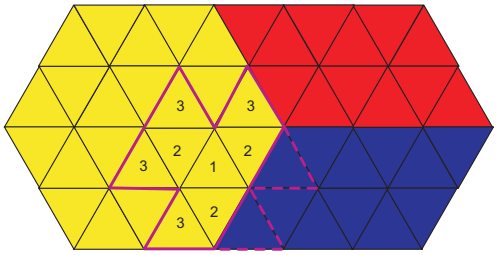

Fig. 1. Iterative control of the material index coding process.

3.2. Texture Attributes Coder

Texture attributes specify the correspondence between polygons on a mesh surface and polygons in a texture image [9], which essentially contain two types of information: texture mapping and texture coordinates. Texture mapping specifies for each vertex whether cuts are introduced when parameterizing its 1-ring neighborhood and, if any, the structure of the cut(s). Texture coordinates specify for each vertex the locations of its incident corners parameterized on the 2D texture image.

As in [8, 9], vertices(corners) are classified as smooth and non-smooth vertices(corners), as shown in Fig. 2 where the corners' texture coordinates are color coded. During the counter-clockwise traversal of the corners around a vertex, if a corner has the same texture coordinates as the previous one, it is called a smooth corner; otherwise, it is a non-smooth corner. The corners c_1 to c_6 are all smooth ones in Fig. 2(a). If all corners around a vertex are smooth ones, that vertex is called a smooth vertex; otherwise, it is a non-smooth vertex. For instance, the central vertex in Fig. 2(a) is a smooth one while the central vertex in Fig. 2(b) is a non-smooth one since c_1 and c_4 are non-smooth corners. For an edge, if, at each end vertex, the corner(s) incident on this edge has(have) the same texture coordinates, this edge is called a smooth edge; otherwise it is a non-smooth edge. For instance, the yellow-colored edge in Fig. 2(c) is a smooth one since c_1 and c_2 have the same texture coordinates and c_3 and c_4 have the same texture coordinates; the yellow-colored edge in Fig. 2(d) is a non-smooth one since c_1 and c_2 have different texture coordinates and so do c_3 and c_4.

Similar to [8, 9], the proposed texture attributes coder is driven by a bread-first traversal of the 3D surface. The baseline coding process is described as follows. Two global queues (initially set to empty) are used in our algorithm, one storing the smooth and the other storing the non-smooth vertices on the boundary of the currently traversed region. We first make a virtual vertex whose sole neighbor is the real vertex with the smallest index, and push this virtual vertex into

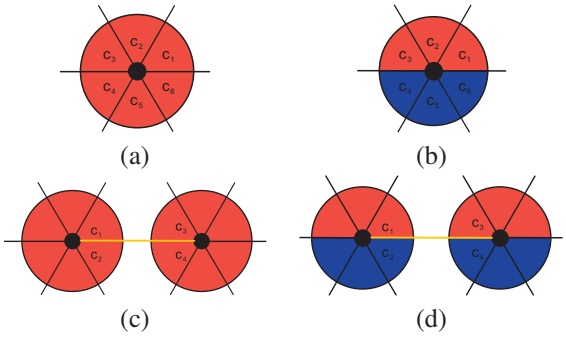

Fig. 2. Examples of (a) a smooth vertex, (b) a non-smooth vertex, (c) a smooth edge, and (d) a non-smooth edge.

the smooth vertex queue. Thereafter, the proposed algorithm runs in iterations. At each iteration, we always take out the first entry from the smooth vertex queue if it is not empty; otherwise, we take out the first one from the non-smooth vertex queue if it is not empty. This entry provides the vertex of focus and we examine all its 1-ring neighbor vertices. For each neighbor, we encode one flag bit (vertex bit) indicating whether it is a smooth one. If it is, we encode its texture coordinates; otherwise, we encode a flag bit (corner bit) for each corner on this vertex indicating whether it is a smooth corner, and encode the texture coordinates for each non-smooth corner. Each 1-ring neighbor of the focus vertex is then pushed to the smooth or non-smooth vertex queue, depending on its type. When both the smooth and the non-smooth vertex queues become empty, we check if all vertices have been traversed. If so, the algorithm terminates; otherwise, we make a virtual vertex whose sole neighbor is the one with the smallest index among all the untraversed real vertices, push this virtual vertex into the smooth vertex queue and repeat the above process. Note that vertex bits, corner bits and texture coordinates are all encoded with arithmetic coders. Compared with [8, 9], the proposed texture attributes coder distinguishes in that it uses two vertex queues and always prioritizes expanding smooth vertices. This facilitates effective texture coordinates prediction as explained below.

To improve the baseline texture attributes coder, we employ effective prediction techniques. For the coding of texture mapping, we make vertex bit and corner bit predictions using the same set of rules as proposed in [8]. Experiments show that the predictive coding can reduce the texture mapping coding bits by around 50%-60% on our test models. For the texture coordinates prediction, however, we propose a novel similar-triangle-based prediction method. Assume that we are to encode the texture coordinates, t_3, for the corner at v_3 in $\triangle v_1v_2v_3$, as shown in Fig. 3. If the texture coordinates ,t_1 and t_2, at v_1 and v_2, respectively, are already encoded, $\langle v_1, v_2\rangle$ is a smooth edge, and the texture coordinates of the corner at v_4 in $\triangle v_2v_1v_4$ are already encoded, the encoder in [9] will predict t_3 to be the 2D position that forms a parallelogram with t_1, t_4 and t_2. However, it does not make use of the 3D model's geometry information (*e.g.*, v_1, v_2, v_3 and v_4) in predicting the texture coordinates. Frequently, surface parameterization is made to preserve angles and/or shapes, leading to high similarity between a triangle on the surface and its parameterization in the texture domain. Therefore, we propose to predict t_3 as a 2D position such that $\triangle t_1t_2t_3$ is similar to $\triangle v_1v_2v_3$. There are two candidate positions, t'_3 and $t"_3$, both satisfying the similarity condition, as shown in Fig. 3. From these two candidates, we select t'_3 as our prediction since it is farther from t_4 than t''_3 and there is no overlapping between $\triangle t_1t_4t_2$ and $\triangle t_1t_2t'_3$.

We see that one necessary condition for the similar-triangle-based prediction is that $\langle v_1, v_2\rangle$ is a smooth edge. Recall that we prioritize smooth vertex expansion during the surface traversal, which exposes smooth edges earlier in the encoding process and therefore increases the number of applications of the similar-triangle-based prediction.

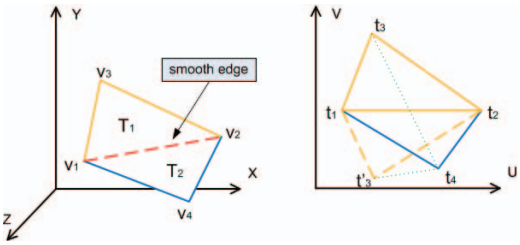

Fig. 3. Illustration of the similar-triangle-based prediction. We are to encode the corner texture coordinates at v_3 inside triangle T_1 whose vertices are v_1, v_2, and v_3. The corresponding corner texture coordinates at v_1 and v_2 are t_1 and t_2, respectively. We have candidate texture coordinates t'_3 and t''_3, each forming a triangle with t_1 and t_2 which is similar to T_1. In order to select the right one, we find another triangle T_2 sharing a common smooth edge $\langle v_1,v_2\rangle$ with T_1. If the corners of T_2 are all encoded and the corner at v_4 has texture coordinates of t_4, we compute the distances from t'_3 and t''_3 to t_4, respectively, and select the one with the longer distance. In this case, we select t'_3 since it is farther from t_4 than t''_3.

Whenever possible, we predict a corner's texture coordinates using the similar triangles prediction rule. When failing to apply this rule, similar to [9], we use delta prediction or corner prediction. When the first corner in a facet is to be encoded, we employ the delta prediction that predicts the current corner's texture coordinates with (0.5, 0.5) corresponding to the center of the texture image; otherwise, we predict the current corner's texture coordinates with those of the encoded corner in the same triangle which is closer to the current.

4. EXPERIMENTAL RESULTS

Test models used in our experiments are shown in Fig. 4. All these models are texture-mapped. NewBld05 and NewBld11 have multiple material attributes defined over the surface, while each of the other four has just one global material defined over the surface. In our experiments, the raw numbers of bits used for the material index are 4 and 6 for NewBld05 and NewBld11, respectively, as determined by the number of material vectors in each. The compression rates in bits per 2-D texture coordinates (bpt), or equivalently bits per corner, are given at three quantization levels: 8, 10 and 12 bits per coordinate. We compare with two representative algorithms for texture attributes compression [8, 9] that were proposed by Martin Isenburg and Jack Snoeyink. Their algorithms will be abbreviated as MIJS in the following text.

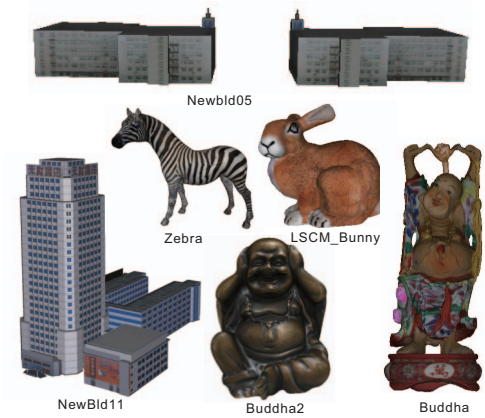

Fig. 4. Models used in our experiments.

Table 1 gives the material index and texture attributes coding bitrates using our method and the texture attributes coding bitrates using MIJS. Note that the benchmark program for MIJS (http://www.cs.unc.edu/∼isenburg/pmc/) does not give coding results for the material attributes, as indicated with "N/A" in the table. Zebra, Buddha, Buddha2 and LSCM_bunny each has just one global material whose encoding cost is negligible, and we use "N/A" to indicate this as well. It is worth mentioning that the raw numbers of material index bits before compression are 2.8 and 4.0 bits per vertex for NewBld05 and NewBld11, respectively. From Table 1, we see that the coding performance of our method is better than or comparable with that of MIJS in terms of total texture attributes coding bitrates (TM+TC) on most of our test models. Particularly, at the quantization level of 12 bits, the advantage of our method over MIJS becomes more obvious. This may be due to the consistent quantization levels (both 12 bits) between the geometry and the texture coordinates, facilitating a more accurate similar-triangle-based prediction. Among the test models, our method gains the most texture coordinate coding advantages over MIJS on Buddha2 and LSCM_bunny. This is due to the fact that Buddha2 and LSCM_bunny are parameterized in a way to preserve angles (*e.g.*, least squares conformal maps for LSCM_bunny) while for angle-preserving parameterization our similarity-triangle-based prediction works the best. Besides yielding good coding bitrates for the texture attributes, another advantage of our method is that it proposes a novel and effective material coding approach while MIJS does not explicitly take care of material coding.

Table 1. Material index and texture attribute coding bitrates for test meshes using our algorithm and MIJS. MA, TM and TC stands for material index coding bitrates, texture mapping coding bitrates and texture coordinate coding bitrates, respectively. Texture coordinate coding bitrates are reported in the unit of bits per 2-D texture coordinates (bpt), while the other two in the unit of bits per vertex (bpv).

Mesh(#v/#vt)	Encoder	MA	TM	TC		
				8bit	10bit	12bit
NewBld05	Ours	1.20	0.02	10.16	12.99	16.46
(442/442)	MIJS	N/A	0.01	10.45	12.85	16.58
NewBld11	Ours	1.63	1.10	7.72	11.72	15.57
(1,489/1,475)	MIJS	N/A	1.62	9.40	11.85	15.24
Zebra	Ours	N/A	0.20	4.89	7.31	10.92
(20,157/21,305)	MIJS	N/A	0.16	4.13	7.47	11.40
Buddha	Ours	N/A	0.61	5.91	8.47	12.01
(25,697/29,609)	MIJS	N/A	0.14	4.74	8.15	12.10
Buddha2	Ours	N/A	0.17	4.84	6.80	10.25
(15,138/15,820)	MIJS	N/A	0.13	4.53	8.05	12.00
LSCM_bunny	Ours	N/A	0.22	4.11	4.48	4.96
(34,834/36,425)	MIJS	N/A	0.14	3.02	4.47	6.75

5. CONCLUSION AND FUTURE WORK

In this work, we have proposed a scheme for compressing material and texture attributes in triangular meshes. For the material coding, breadth-first surface traversal based methods are proposed to exploit the local coherence of the material attributes. Further, prioritized surface traversal and effective prediction method are proposed for texture attributes coding. As a result, the proposed scheme leads to better or comparable coding performance at various quantization levels on our test models.

We currently work on single-rate compression of material and texture attributes. In the future, it is worthwhile to investigate how to compute and encode the material and texture attributes in combination with progressive 3D model compression.

Acknowledgment

This research was supported by Grant No. 2010CB328002 from The National Basic Research Program of China (973 Program), Grant No. 61232014, 61121002, 61173080, 61303083 and U1035004 from National Natural Science Foundation of China, Grant No. ZR2011FZ004 from Shandong Provincial Natural Science Foundation, China, and the Program for New Century Excellent Talents in University (NCET) in China.

6. REFERENCES

[1] J. Peng, C.-S. Kim, and C.-C. J. Kuo, "Technologies for 3D mesh compression: A survey," *Journal of Visual Communication and Image Representation*, vol. 16, no. 6, pp. 688–733, 2005.

[2] P. Alliez and C. Gotsman, "Recent advances in compression of 3d meshes," in *Proc. of the Symp. on Multiresolution in Geometric Modeling*, Sep 2003.

[3] C. Gotsman, S. Gumhold, and L. Kobbelt, "Simplification and compression of 3D meshes," in *Tutorials on Multiresolution in Geometric Modelling*, 2002.

[4] M. Deering, "Geometry compression," in *ACM SIGGRAPH*, 1995, pp. 13–20.

[5] G. Taubin and J. Rossignac, "Geometric compression through topological surgery," *ACM Trans. Graphics*, vol. 17, no. 2, pp. 84–115, 1998.

[6] Chandrajit L Bajaj, Valerio Pascucci, and Guozhong Zhuang, "Single-resolution compression of arbitrary triangular meshes with properties," in *Data Compression Conference, 1999. Proceedings. DCC'99*. IEEE, 1999, pp. 247–256.

[7] Martin Isenburg and Jack Snoeyink, "Face fixer: Compressing polygon meshes with properties," in *ACM SIGGRAPH*, 2000, pp. 263–270.

[8] Martin Isenburg and Jack Snoeyink, "Compressing the property mapping of polygon meshes," in *Proceedings of Pacific Graphics*, 2001, pp. 4–11.

[9] Martin Isenburg and Jack Snoeyink, "Compressing texture coordinates with selective linear predictions," in *Proceedings of Computer Graphics International*, 2003, pp. 126–131.

[10] Jeong hwan Ahn, Chang su Kim, and Yo sung Ho, "Predictive compression of geometry, color and normal data of 3-D mesh models," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, pp. 291–299, 2006.

[11] Young suk Yoon, Sung yeol Kim, and Yo sung Ho, "Color Data Coding for Three-Dimensional Mesh Models Considering Connectivity and Geometry Information," in *International Conference on Multimedia Computing and Systems/International Conference on Multimedia and Expo*, 2006, pp. 253–256.