# HYBRID COMPRESSION OF DYNAMIC 3D MESH DATA

*Choong-Hoon Kwak and Ivan V. Bajić*

School of Engineering Science, Simon Fraser University, Burnaby, BC, Canada

## ABSTRACT

Geometric representation of objects and surfaces in terms of 3D meshes is becoming increasingly important in a variety of applications. As the amount of data in this format increases, the problem of compression becomes vital for the further development of the field. In this paper we present a codec for dynamic 3D mesh data that utilizes the "hybrid" framework from video coding, built upon temporal prediction and spatial transform. We discuss various features of the codec, including unrestricted quantization and two-stage entropy coding, and investigate its compression efficiency on a variety of test material. A discussion of various prediction structures and their impact on error resilience is also provided.

***Index Terms***— Dynamic geometry compression, 3D mesh compression, unrestricted quantization

## 1. INTRODUCTION

3D graphics data are increasingly used in a number of applications. 3D Computer Aided Design (CAD) is replacing traditional architectural design [1], and has long been used in vehicle and aircraft industry. Clinical applications, such as orthodontic diagnosis using 3D scans [2], are also on the rise. 3D scientific data visualization, for example in weather modeling [3], enrich our understanding of the underlying phenomena and enable better analysis. In addition, the use of 3D data has been popular in the entertainment industry, especially video gaming and film.

Early work on geometry compression was focused on static data, while current efforts mainly consider dynamic data. In [4], a compression technique was constructed using Principal Component Analysis (PCA) applied over all vertices of all frames in the sequence. In [5], PCA is used along with second-order linear prediction, while in [6], a clustering-based PCA compression is proposed. Vertices are segmented into clusters and PCA is applied to each cluster separately. In [7], PCA is applied along with EdgeBreaker techniques to compress 3D mesh data.

In [8], a coding method based on predictive Discrete Cosine Transform (DCT) is proposed. The encoder clusters the vertices with similar motions and applies DCT to the difference of consecutive frames in each clusters. A coding method based on key frames is introduced in [9], where the encoder selects and encodes certain key frames, while the decoder reconstructs these key frames and interpolates intermediate frames. In [10], an octree-based coding method is proposed. Octree is an efficient structure for encoding 3D data, and is applied in [10] to 3D motion vectors of mesh vertices. This method has been extended in [11] through rate-distortion optimization to improve coding efficiency.

Frame-based Animated Mesh Compression (FAMC) [12] is a technique adopted in MPEG-4 part 16 AFX (Animation Framework eXtension). It consists of spatial clustering of vertices to the point where the motion within each cluster can be approximated by a single 3D affine motion model. The first frame is encoded using static mesh coding techniques, while skinning-based motion prediction is applied to subsequent frames. Prediction residuals are encoded with the help of a temporal transform, e.g. DCT. FAMC framework was extended in [13] in terms of spatial and temporal scalability.

In this paper, our earlier Motion Capture (MoCap) data codec [14, 15] is adapted to the case of dynamic 3D mesh data. The distinguishing features of our approach compared to those listed above is the avoidance of temporal transforms, which enables low-delay encoding, and avoidance of spatial clustering, which simplifies encoding process and does not require cluster updates. The paper is organized as follows. The coding process is described in Section 2, with quantization discussed in Section 2.3, entropy coding in Section 2.4, and prediction structures in Section 2.5. Coding performance on several 3D mesh sequences is presented in Section 3, followed by conclusions in Section 4.

## 2. HYBRID DYNAMIC 3D MESH CODING

3D mesh data consists of two parts: (1) 3D geometry, i.e., coordinates of the vertices, and (2) mesh connectivity information, which specifies which vertices form mesh faces. Connectivity data often does not change throughout the sequence, at least that was the case with the data used for evaluating the proposed codec.[1] The connectivity data was simply encoded in a lossless manner in the first frame using a version of adaptive arithmetic coding known as range coding [16], and reused

This work was supported in part by the NSERC Grant 430592.

[1] Evaluation was carried out in accordance with the rules of the first IEEE SPS Dynamic Geometry Compression Competition: http://www.geometrycompression.org/

978-1-4799-2893-4/14/$31.00 ©2014 IEEE

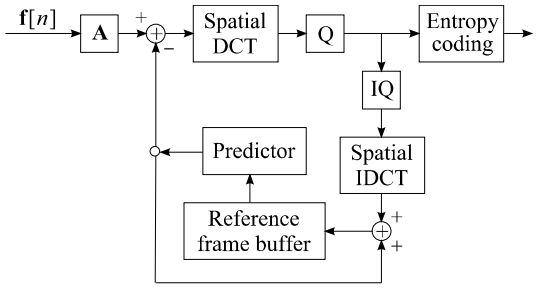


**Fig. 1**: Block diagram of a hybrid dynamic mesh encoder.

in all subsequent frames to reconstruct the mesh. It should be noted that there are several efficient methods for encoding connectivity data, e.g. [17, 18], which could be used instead for improved efficiency. However, if connectivity is constant across frames, differences in coding efficiency of connectivity data do not play a major role in long sequences. In the remainder, we focus on lossy compression of 3D geometry.

The structure of the proposed hybrid 3D mesh encoder is shown in Fig. 1 and follows the structure of MoCap encoders from [14, 15]. While MoCap data has a much lower spatial sampling density compared to 3D mesh data (i.e., a few tens of motion markers compared to a few thousand mesh vertices), the techniques used in MoCap coding turn out to be useful in 3D mesh coding as well. The components of the encoder are briefly described below.

### 2.1. Reordering

The input to the encoder is a frame $\mathbf{f}[n]$ of 3D vertex coordinates. These coordinates are usually grouped according to the vertex, that is, 3D coordinates of the first vertex, followed by 3D coordinates of the second vertex, and so on. Within the coding loop, prediction residuals of the vertices will be subject to the spatial transform, so it would be beneficial if similar coordinates were grouped together. That way, the transform would be better able to exploit the correlation among the coordinates. One simple way to achieve this is to reorder the coordinates according to the axis, creating a sequence in which x-coordinates of all vertices appear first, followed by y-coordinates and z-coordinates. As discussed in [14], such reordering (matrix $\mathbf{A}$ in Fig. 1) helps to concentrate the signal energy at low frequencies. After quantization, this results in a number of consecutive zeros at high frequencies, which facilitates efficient entropy coding.

### 2.2. Spatial transform

Upon reordering, prediction of the current input vector is subtracted from the actual input vector, and the residual is subject to spatial transform. Our codec applies 1D DCT (block size equal to the prediction vector length) to the entire prediction residual vector. Spatial transform exploits spatial redundancy in the data to concentrate signal energy into relatively few low-frequency coefficients. It plays a role similar to spatial clustering in FAMC [12] and several other mesh encoders, but is much simpler.

### 2.3. Quantization

Unlike image and video data whose bit depth, and therefore dynamic range, is known in advance, the dynamic range of 3D mesh data depends on various factors such as the choice of the origin, and the size and dynamics of the mesh (e.g., how much it increases or moves relative to the origin). One can of course make assumptions about the dynamic range in advance, but large overload distortion will result when such assumptions fail. Another option is to analyze the whole mesh sequence prior to encoding and empirically find the dynamic range, but this is only suitable for offline encoding. To avoid such problems, our encoder employs an unrestricted midtread quantizer [15], which maps an input value $x$ into the quantization index $q$ given by

$$q = \mathcal{Q}(x) = \mathrm{sgn}(x)\lfloor |x|/\Delta + 0.5 \rfloor, \qquad (1)$$

where $\Delta$ is the quantizer step size, $\mathrm{sgn}(x)$ is the sign of $x$, and $\lfloor \cdot \rfloor$ denotes rounding-down operation. Dequantized value is given by $\widehat{x} = q\Delta$. Compared to conventional quantizers that divide a finite input range into a finite number of bins, thereby creating a possibility for large overload distortion in the outermost bins, the unrestricted quantizer divides the entire real line into bins of size $\Delta$ and therefore eliminates the possibility of overload distortion. While this allows one to guarantee $|x - \widehat{x}| \leq \Delta/2$ for any $x \in \mathbb{R}$, the downside is that there are now an infinite number of quantization indices $q$, which prevents conventional entropy coding techniques such as Huffman or arithmetic coding from being used directly on the output of the quantizer.

### 2.4. Entropy coding

Since the number of possible quantizer outputs $q$ is infinite, the following strategy is employed to encode them efficiently [15]. The first bit in the bitstream of any frame indicates whether all any quantization indices in the that frame are non-zero. When there is no motion in the sequence, or at very low bitrates when the quantizer step size is large, it sometimes happens that all residuals are quantized to zero, and this strategy provides an effective way of encoding such frames by a single bit. Otherwise, if there are non-zero indices in the frame, the following procedure is utilized to encode them, starting from those corresponding to low-frequency DCT coefficients.

The quantization index $q$ is separated into sign $\mathrm{sgn}(q)$ and magnitude $|q|$. The magnitude is encoded using adaptive Golomb-Rice coding [19] followed by adaptive range coding.

The adaptive Golomb-Rice encoder computes its parameter $k$ as

$$k = \max\left\{0, \lceil \log_2 \left(0.5 \cdot \text{avg}(\widehat{\mathbf{x}_r})\right) \rceil\right\}, \qquad (2)$$

where $\widehat{\mathbf{x}_r}$ is the vector of de-quantized DCT coefficients in the reference frame, and then sets its divisor to $m = 2^k$. The magnitude is divided as $|q| = mn + r$, where $n$ is the quotient and $0 \leq r < m = 2^k$ is the remainder. The quotient is represented as $n$ ones followed by a zero, while the remainder is represented as a $k$-bit fixed-length codeword. Note that $k$ does not have to be encoded, because the decoder can compute it using (2). After encoding the last non-zero index in the frame, the encoder inserts the End-of-Frame (EOF) symbol and avoids encoding the remaining zero indices. At the decoder, when an EOF symbol is detected, any still-to-be-decoded indices in that frame are set to zero. In summary, the Golomb-Rice encoder binarizes the quantization index magnitudes and places the EOF symbol after the last non-zero index, thereby eliminating the need to code a potentially large number of trailing zeros at high frequencies.

The adaptive range encoder uses three symbols from the output of the Golomb-Rice encoder: ones and zeros that represent binarized index magnitudes, and the EOF symbol. The role of arithmetic coder is to improve coding efficiency of Golomb-Rice coding that may result if the empirical distribution of quantization indices is not geometric (in which case Golomb-Rice code would be optimal). All symbol frequencies are initialized to uniform at the beginning of each frame and are updated as the encoding proceeds. This is one of the differences with respect to the MoCap encoder in [15], where, due to the relatively small number of samples per frame, symbol frequencies were carried from frame to frame in order to allow more effective adaptation. In the case of 3D mesh data, with thousands of vertices per frame, such carry over was found not to be necessary. Another difference with respect to [15] is rescaling the symbol counts if they reach $2^{16} - 1$, to avoid counter overflow. Whenever this occurs, the symbol counts of zeros and ones are divided by 2 and rounded down, while the count of EOF symbol is left as is, since there is only one such symbol per frame. This rescaling was not necessary in the MoCap codec [15] due to the relatively small number of samples per frame. The overall structure of the bitstream for a single frame is shown in Fig. 2. Sign bits follow the magnitude bitstream and are left uncoded since their distribution is close to i.i.d., and therefore not very compressible.

It should be noted that the combination of Golomb coding and arithmetic coding was also used for 3D mesh compression in [20]. However, the procedure employed in [20] was to arithmetically encode quantized residuals between $-3$ and 3, and Golomb-code the residuals outside of this range. Unlike [20], our encoder uses adaptive Golomb-Rice encoding to binarize all quantized residuals, and then arithmetically encodes this binarized stream.

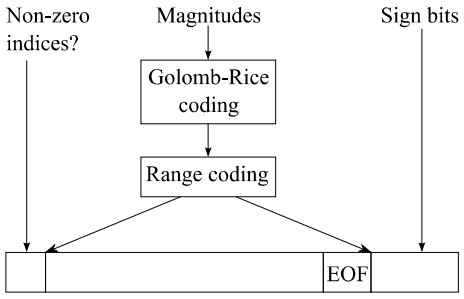


**Fig. 2**: Structure of the encoded bitstream.

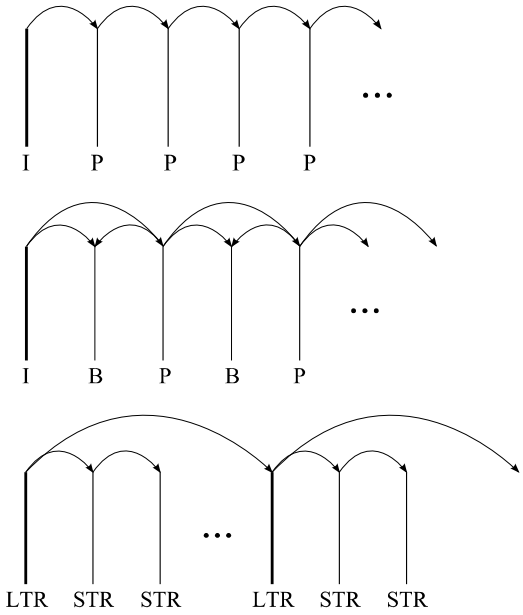


**Fig. 3**: Various possible prediction structures.

### 2.5. Prediction

The encoder architecture in Fig. 1 offers a flexible framework in which various prediction structures can be employed, similar to hybrid video encoders. Some of these are shown in Fig. 3. The top one is the well known IPP structure from video coding, in which the first frame (I) is intra-coded without prediction from any other frame, and subsequent frames are predictively (P) encoded from reconstructed previous frames. Various predictors (e.g., zero-order, first-order, etc.) can be utilized; in our experiments, we used a simple zero-order predictor. The IPP structure has zero-frame algorithmic delay, meaning that the mesh data in any frame can be encoded as soon the frame is captured, and is therefore very suitable for interactive applications.

The second configuration in Fig. 3 is the IBP structure in which, in addition to I and P frames, the bi-directionally (B) predicted frames are utilized. Although only one B frame is shown between neighboring I/P frames, the number of consecutive B frames can be larger, as in video coding. This

structure offers higher compression efficiency at the cost of increased algorithmic delay, because B frames cannot be encoded until their future reference frame is encoded.

Conventional IPP and IBP structures are vulnerable to potential errors and losses that may occur when the bitstream is transmitted over a noisy channel. The last structure in Fig. 3 takes error propagation into account and attempts to minimize its effects at the cost of some degradation in compression efficiency. This was the prediction structure employed in our original MoCap codec [14, 15]. It contains two types of frames - Long-Term Reference (LTR) frames, which are predicted only from the previous LTR frame, and Short-Term Reference (STR) frames, which are simply predicted from the previous frame (either LTR or STR), and act like P frames in the IPP structure. The LTR/STR structure has zero-frame algorithmic delay, yet is more error resilient than the IPP structure, because any errors in the STR frames cannot propagate beyond the next LTR frame. Error concealment for this prediction structure was developed in [21]. In video coding, error resilient prediction structures have been studied in [22, 23, 24], among others, and some of these techniques may be applicable to error resilient mesh prediction as well.

## 3. EXPERIMENTAL RESULTS

Experiments were carried out on five test sequences from [25]: *Dance*, *Dog*, *Handstand*, *Skirt*, and *Wheel*. Three codecs were tested. One is our earlier Hybrid Motion Capture Codec (HMOCC) [15], in which only LTR frames are utilized (i.e., there are no STR frames, so LTR frames act simply as P frames). The other two codecs are two versions of the Hybrid Dynamic Mesh Compression (HDMC) described in this paper, one employing the IPP prediction structure (HDMC-IPP) and the other employing the IBP structure (HDMC-IBP). Compression efficiency was measured in terms of Root Mean Square (RMS) error versus total file size.

Operational rate-distortion curves of the three codecs are shown in Fig. 4. As expected, HDMC-IBP has the best performance on all sequences, followed by HDMC-IPP and HMOCC. A more quantitative comparison is provided in Table 1, which shows the Bjontegaard Delta (BD) [26] performance of HDMC-IPP and HDMC-IBP relative to HMOCC. As seen in the table, HDMC-IPP offers up to 11% bit savings compared to HMOCC, while HHDMC-IBP is able to provide up to 24% bit savings.

## 4. DISCUSSION AND CONCLUSIONS

In this paper, our earlier Motion Capture codec [15] was modified and employed for dynamic 3D mesh compression. The modified codec with the IBP prediction structure was able to achieve up to 24% bit savings relative to the original MoCap codec. Although rate-distortion comparison with conventional 3D mesh codecs was not carried out in this paper,

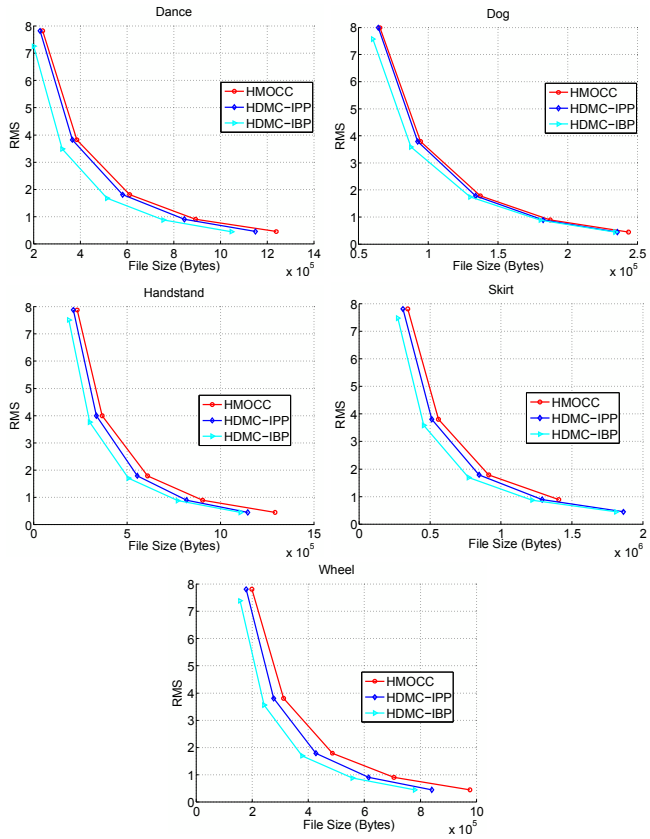


**Fig. 4**: RMS vs. file size on five test sequences.

architectural differences suggest that the proposed codec is simpler and offers a flexible framework able to accommodate both low-delay compression as well as error resilience.

Further enhancements to the presented codec are possible in several ways. Quantization step size, which was kept constant in the present work for simplicity, may be adapted based on the content, type of frame, or perceptual importance of various coefficients of the spatial DCT, in order to achieve desired bit allocation or rate control. Other spatial transforms (e.g., a graph transform) could be employed instead of DCT. Finally, context-adaptive entropy coding and more advanced predictors, such as the skinning-based motion compensation from FAMC [12], could be incorporated into the framework.

| | BD RMS | | BD file size (%) | |
|---|---|---|---|---|
| **Sequence** | IPP | IBP | IPP | IBP |
| *Dance* | −0.22 | −0.93 | −5.29 | −20.02 |
| *Dog* | −0.13 | −0.48 | −2.41 | −9.75 |
| *Handstand* | −0.40 | −0.90 | −9.17 | −20.29 |
| *Skirt* | −0.40 | −0.95 | −7.04 | −18.90 |
| *Wheel* | −0.56 | −1.18 | −11.55 | −24.20 |

**Table 1**: BD performance relative to HMOCC.

## 5. REFERENCES


[1] J. J. Van Wijk, B. de Vries, and C. W. A. M. van Overveld, "Towards an understanding 3D VR architectural design system," in *Interactions in Virtual Worlds*, 1999, pp. 219–224.

[2] M. Y. Hajeer, D. T. Millett, A. F. Ayoub, and J. P. Siebert, "Current products and practices applications of 3D imaging in orthodontics: Part II," *J. Orthod.*, vol. 31, no. 2, pp. 154–162, 2004.

[3] P. Ramachandran and G. Varoquaux, "Mayavi: 3D visualization of scientific data," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 40–51, 2011.

[4] M. Alexa and W. Muller, "Representing animations by principal components," *Computer Graphics Forum*, vol. 19, no. 3, pp. 411–418, 2000.

[5] Z. Karni and C. Gotsman, "Compression of soft-body animation sequences," *Computers & Graphics*, vol. 28, no. 1, pp. 25–34, 2004.

[6] M. Sattler, R. Sarlette, and R. Klein, "Simple and efficient compression of animation sequences," in *Proc. ACM SIGGRAPH/Eurographics Symp. Computer Animation*, 2005, pp. 209–217.

[7] L. Vasa and V. Skala, "Coddyac: Connectivity driven dynamic mesh compression," in *Proc. IEEE 3DTV Conf.* IEEE, 2007.

[8] R. Amjoun and W. Strasser, "Predictive-DCT coding for 3D mesh sequences compression," *J. Virtual Reality and Broadcasting*, vol. 5, no. 6, 2008.

[9] E. S. Jang, J. D. K. Kim, S. Y. Jung, M.-J. Han, S. O. Woo, and S.-J. Lee, "Interpolator data compression for MPEG-4 animation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 14, no. 7, pp. 989–1008, 2004.

[10] J. Zhang and C. B. Owen, "Octree-based animated geometry compression," *Computers & Graphics*, vol. 31, no. 3, pp. 463–479, 2007.

[11] K. Müller, A. Smolic, M. Kautzner, P. Eisert, and T. Wiegand, "Rate-distortion-optimized predictive compression of dynamic 3D mesh sequences," *Signal Processing: Image Communication*, vol. 21, no. 9, pp. 812–828, 2006.

[12] K. Mamou, T. Zaharia, and F. Preteux, "FAMC: The MPEG-4 standard for animated mesh compression," in *Proc. IEEE ICIP'08*, 2008, pp. 2676–2679.

[13] N. Stefanoski and J. Ostermann, "Spatially and temporally scalable compression of animated 3D meshes with MPEG-4 / FAMC," in *Proc. IEEE ICIP'08*, 2008, pp. 2696–2699.

[14] C. H. Kwak and I. V. Bajić, "Hybrid low-delay compression of motion capture data," in *Proc. IEEE ICME'11*, Barcelona, Spain, July 2011.

[15] C. H. Kwak and I. V. Bajić, "MoCap data coding with unrestricted quantization and rate control," in *Proc. IEEE ICASSP'13*, Vancouver, BC, May 2013, pp. 3741–3745.

[16] M. Servais, *Range Coding in MATLAB*, [Online]. Available: http://www.ee.surrey.ac.uk/CVSSP/VMRG/hdtv/code.htm.

[17] C. Touma and C. Gotsman, "Triangle mesh compression," in *Graphics Interface*, 1998, vol. 98, pp. 26–34.

[18] J. Rossignac, "Edgebreaker: Connectivity compression for triangle meshes," *IEEE Trans. Vis. Comput. Graphics*, vol. 5, no. 1, pp. 47–61, 1999.

[19] N. Memon, "Adaptive coding of DCT coefficients by Golomb-Rice codes," in *Proc. IEEE ICIP'98*, Chicago, IL, 1998, vol. 1, pp. 516–520.

[20] N. Stefanoski and J. Ostermann, "Connectivity-guided predictive compression of dynamic 3D meshes," in *Proc. IEEE ICIP'06*, 2006, pp. 2973–2976.

[21] C. H. Kwak and I. V. Bajić, "Error concealment strategies for motion capture data streaming," in *Proc. IEEE ICME'11 Workshops - StreamComm*, Barcelona, Spain, July 2011.

[22] Y. J. Liang, M. Flierl, and B. Girod, "Low-latency video transmission over lossy packet networks using rate-distortion optimized reference picture selection," in *Proc. IEEE ICIP'02*, Rochester, NY, 2002, vol. 2, pp. 181–184.

[23] G. Cheung, W.-t. Tan, and C. Chan, "Reference frame optimization for multiple-path video streaming with complexity scaling," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 6, pp. 649–662, June 2007.

[24] S. M. Amiri and I. V. Bajić, "A two-stage H.264/AVC encoder for video streaming with fast reference picture selection," in *Proc. ACM WMuNeP'08*, Vancouver, BC, Oct. 2008, pp. 37–44.

[25] J. Gall, C. Stoll, E. De Aguiar, C. Theobalt, B. Rosenhahn, and H.-P. Seidel, "Motion capture using joint skeleton tracking and surface estimation," in *Proc. IEEE CVPR'09*, 2009, pp. 1746–1753.

[26] G. Bjontegaard, "Calculation of average PSNR differences between RD-curves," *ITU-T VCEG-M33*, 2001.