POINT CLOUD ATTRIBUTE COMPRESSION WITH GRAPH TRANSFORM

Cha Zhang, Dinei Florêncio and Charles Loop

Microsoft Research
One Microsoft Way, Redmond, WA 98075

ABSTRACT

Compressing attributes on 3D point clouds such as colors or normal directions has been a challenging problem, since these attribute signals are unstructured. In this paper, we propose to compress such attributes with graph transform. We construct graphs on small neighborhoods of the point cloud by connecting nearby points, and treat the attributes as signals over the graph. The graph transform, which is equivalent to Karhunen-Loève Transform on such graphs, is then adopted to decorrelate the signal. Experimental results on a number of point clouds representing human upper bodies demonstrate that our method is much more efficient than traditional schemes such as octree-based methods.

Index Terms— Graph transform, compression, 3D point cloud, 3D voxel model

1. INTRODUCTION

Capturing 3D models for real-world scenes has been an active research topic for decades, with wide applications in gaming, telepresence, heritage preservation, etc. Thanks to the commoditization of 3D depth sensors, it has become easy to digitize the world into 3D models with millions of points. Such point cloud data usually occupy a large amounts of storage space, and demand efficient compression algorithms to store and transmit effectively.

One possibility is to convert the point clouds into polygonal meshes, which can be compressed with a large body of existing methods [1]. Polygonal meshes are easy to edit and ideal for synthetic 3D objects, which have been used extensively in gaming. However, the conversion process is usually computationally expensive, thus making it difficult to be applied in real-time applications such as telepresence.

Consequently, many researchers have opted to compress the point cloud directly. For instance, in their renowned QSplat paper, Rusinkiewicz and Levoy [2] used a compact representation of 48 bits to encode the position, normal and color attributes of each 3D point. Ochotta and Saupe [3] partition the point set into clusters and project them onto a regular grid, such that the corresponding height field can be encoded using 2D wavelet transforms. In [4] and [5], the authors propose to compress the point cloud with an octree decomposition of 3D space. The child cell configurations are further coded predictively based on a local surface estimation. More recently, Jiang *et al.* [6] further improved cell configuration prediction based on tangent-plane-continuity maximization and hierarchical probability estimation of occupancy codes. Kammerl *et al.* [7] present a technique for comparing the octree data structures of consecutive point clouds, thus enabling temporal prediction between point cloud frames in a teleoperation application.

Despite the many efforts in point cloud compression, most of them focus heavily on the efficiency of coding the 3D point positions instead of their attributes (e.g., colors and normals). These attributes are critical in rendering the point cloud with high quality. However, unlike traditional images and videos, these attributes lie on a completely unstructured point cloud, and are thus difficult to compress. Recent work [5] encodes color with a linear de-correlation transform followed by adaptive quantization along each transformed axis. While this scheme is fast, it can only achieve coding efficiency similar to an octree-based method.

In this paper, we propose to apply graph transform on the compression of point cloud attributes. We assume the 3D positions of the point cloud have already been coded, and form an octree data structure. We then traverse the octree and construct a graph for each branch of leaves at certain levels of the octree. Graph transform is then applied to the attributes in order to compress them efficiently. Experimental results on a number of point clouds representing human upper bodies demonstrate that our method is much more efficient than traditional schemes, such as octree-based compression.

The rest of the paper is organized as follows. Section 2 first gives an introduction to graph transform, and then explains its application on point clouds. The quantization and entropy coding of the transformed coefficients require some special treatment, and are explained in Section 3. Experimental results and conclusions are given in Section 4 and 5, respectively.

2. GRAPH TRANSFORM

2.1. Background

Graphs are generic data representations that can be used to describe signals in numerous applications, including social networks, energy, transportation, etc. Graph signal processing has attracted a lot of research interest recently [8][9]. In this paper, we consider one particular technique called graph transform. Graph transform is a linear transform determined by the underlying graph structure, and has been shown to be very useful in compressing certain types of signals, such as mesh geometry [10][11], depth maps [12][13][14], or images/videos [15][16]. In the following, we present a very brief introduction on graph transform based on the analysis in [15].

Start with a generic graph $\mathcal{G} = (\mathcal{V} = \{1, \cdots, n\}, \mathcal{E})$, where $\mathcal{V}$ represents the set of nodes in the graph, and $\mathcal{E}$ represents the set of edges. Let us define a multivariate Gaussian random vector $\mathbf{x} = (x_1, \cdots, x_n)^T$ on the graph nodes, often referred as a Gaussian Markov Random Field (GMRF), with mean μ and precision matrix $\mathbf{Q} \geq 0$ (semi-positive definite). Its probability density is:

$$p(\mathbf{x}) = (2\pi)^{-\frac{n}{2}} |\mathbf{Q}|^{\frac{1}{2}} \exp\Big(-\frac{1}{2}(\mathbf{x}-\mu)^T \mathbf{Q}(\mathbf{x}-\mu)\Big), \quad (1)$$

$$\text{and } Q_{ij} \neq 0 \Leftrightarrow \{i,j\} \in \mathcal{E} \text{ for all } i \neq j. \quad (2)$$

Here the precision matrix $\mathbf{Q}$ is the inverse of the covariance matrix $\mathbf{\Sigma}$ in a typical multivariate Gaussian distribution. The above repre-

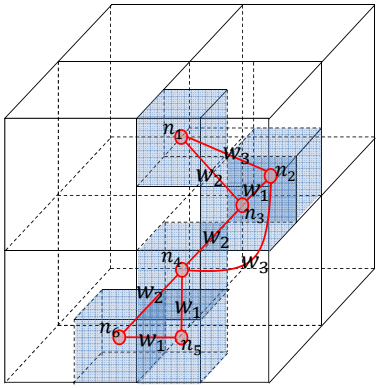

Fig. 1. Forming a graph based on point clouds on an octree block. Here $n_1, \cdots, n_6$ are the nodes of the graph, and $w_1, \cdots, w_3$ are weight values defined on the graph edges.

sentation includes degenerated GMRFs where the determinant of $\mathbf{Q}$ can be zero. Partial correlations between variables can be directly obtained from $\mathbf{Q}$, since [17]:

$$\rho(x_i, x_j | \mathbf{x} \setminus \{x_i, x_j\}) = -\frac{Q_{ij}}{\sqrt{Q_{ii}Q_{jj}}}, i \neq j, \tag{3}$$

where $\rho(x_i, x_j | \mathbf{x} \setminus \{x_i, x_j\})$ represents the partial correlation between x_i and x_j given all other variables.

It is not difficult to show that the eigenvector matrix of the precision matrix is in fact identical to the Karhunen-Loève Transform (KLT) [15], which is optimal for decorrelating the input signal $\mathbf{x}$. Since the above analysis is conducted for a signal residing on a graph, we denote the transform *graph transform*.

In practice, provided a graph with correlations between nodes, one could always define a precision matrix $\mathbf{Q}$, thus implicitly assuming a GMRF model over the graph. Afterwards, form a linear transform matrix by stacking $\mathbf{Q}$'s eigenvectors, and apply the transform matrix on the graph signal. Similar to KLT, graph transform can compact the energy of the signal very effectively, making it ideal for signal compression.

2.2. Graph Transform for Point Cloud Attributes

We assume that the point cloud has already been organized into an octree, where each occupied leaf node is a voxel that represents a point from the point cloud. We choose an octree since there have been many existing approaches [4][5][6][7] that adopt this data structure; although it is not a necessary condition for the proposed method to work. We further assume that the octree is encoded with a certain compression scheme, which can be successfully reconstructed at the decoder. Our goal is to compress the attributes on the point cloud, such as colors and normals. Without loss of generality, we use color attributes as an example in this paper.

Consider a small branch of the octree at a certain level representing, for example, a block containing $k \times k \times k$ voxels. Among these voxels, N may be occupied by the point cloud. Fig. 1 shows such a block with $4 \times 4 \times 4$ voxels and $N = 6$ occupied ones (labeled as shaded cubes). We can form a graph by connecting nearby occupied voxels, and assigning a weight to each edge of the graph. For instance, if we only consider connecting voxel nodes with their 26 neighbors that have a maximum distance of 1 along any axis, we can form a simple graph as in Fig. 1. The weights on the graph shall describe the attribute similarity between nodes, and we set them as inversely proportional to the distances between voxels, i.e.,

$$w_1 = 1, w_2 = \frac{1}{\sqrt{2}}, w_3 = \frac{1}{\sqrt{3}}. \tag{4}$$

An adjacency matrix can be defined from the graph as:

$$\mathbf{A} = \begin{pmatrix} 0 & w_3 & w_2 & 0 & 0 & 0 \\ w_3 & 0 & w_1 & w_3 & 0 & 0 \\ w_2 & w_1 & 0 & w_2 & 0 & 0 \\ 0 & w_3 & w_2 & 0 & w_1 & w_2 \\ 0 & 0 & 0 & w_1 & 0 & w_1 \\ 0 & 0 & 0 & w_2 & w_1 & 0 \end{pmatrix} \tag{5}$$

Let $\mathbf{D} = diag(d_1, \cdots, d_6)$ be a diagonal matrix, whose elements $d_i = \sum_j a_{ij}$, where a_{ij} are the elements in adjacency matrix $\mathbf{A}$. We may then construct a precision matrix $\mathbf{Q}$ of the graph as:

$$\mathbf{Q} = \delta(\mathbf{D} - \mathbf{A}), \tag{6}$$

where δ is a scaler that is related to the graph signal's variance. Note $\mathbf{Q}$ is guaranteed to be positive semi-definite, and is often referred as the non-normalized graph Laplacian.

There can be many different ways to construct the graph and choose the edge weights on a given point cloud, even without the overlayed octree structure. The above example works reasonably well in our experiments and resembles some existing work in the literature [12]. A better graph and its weights can certainly be constructed, e.g., by collecting statistics from many training point clouds. However, given the variety of topologies, it may require an extremely large number of training examples in order to be statistically meaningful.

Consider the eigenvalue decompoistion of $\mathbf{Q}$:

$$\mathbf{Q} = \mathbf{\Phi}\mathbf{\Lambda}\mathbf{\Phi}^{-1}, \tag{7}$$

where $\mathbf{\Lambda} = diag(\lambda_1, \cdots, \lambda_6)$ is a diagonal matrix containing $\mathbf{Q}$'s eigenvalues. The eigenvector matrix $\mathbf{\Phi}$ of the precision matrix $\mathbf{Q}$ can then be used to transform the color signal defined on the graph nodes. This transform is optimum for a signal following the underlying GMRF model with precision matrix $\mathbf{Q}$ [15]. Furthermore, the associated eigenvalue of each eigenvector corresponds to the inverse of the expected variance of the associated transform coefficient (we'll use this in Section 3).

We stress again that the octree structure is encoded and decoded prior to – and separately from – the color information, e.g., using any of the schemes presented in [4][5][6][7]. Thus, both the encoder and decoder can form the same precision matrix for each block of the octree, and no overhead is incurred.

3. QUANTIZATION AND ENTROPY CODING

For each block, the $N \times N$ Graph Transform $\mathbf{\Phi}$ is computed based on the occupancy of the voxels, as defined in Eqs. (4)-(7). Afterwards, the (color) signal residing on these N nodes is made into three $N \times 1$ vectors (corresponding to the YUV components). Each of the three vectors (YUV) is then encoded separately. Let us denote the vector derived from the Y component as $\mathbf{y}$. This vector's transform for the block is computed as $\mathbf{f} = \mathbf{\Phi}\mathbf{y}$, and quantized as $\mathbf{f}_q = round(\mathbf{f}/Q)$, where Q is the quantization step size. The quantized vector is then entropy coded. After all blocks are encoded for the Y component, U and V components are encoded similarly.

bitrate(bpv)	SNR_Y(dB)	SNR_U(dB)	SNR_V (dB)
PCL encoder			
14.15	52.0	54.6	54.5
11.34	44.3	51.2	50.8
8.40	38.0	47.0	46.3
5.70	32.1	41.9	41.4
3.30	26.9	37.7	36.9
1.48	23.0	34.0	33.3
Proposed Graph Transform encoder			
5.36	52.1	54.7	54.6
1.74	44.3	51.4	50.8
0.36	38.1	47.1	46.4
0.16	28.4	38.2	38.2

Table 1. Comparison between our algorithm and the PCL encoder for color data. Distortion (in dB) is measured as the average over 6 point clouds for Y, U, and V.

Because of the way $\mathbf{\Phi}$ is constructed, the first, and possibly a few more coefficients of $\mathbf{f}$ of each block correspond to DC terms. For instance, if the block has m disconnected subsets of points, the first m coefficients in $\mathbf{f}$ will correspond to the average of each of the the connected sub-regions, multiplied by the square root of the number of voxels in the sub-region. Additionally, the eigenvalues associated with all these coefficients will be zero. The rest of the coefficients in $\mathbf{f}$ are AC terms. We treat DC and AC terms differently during entropy coding, as explained below.

3.1. Entropy coding the AC terms

We use a simple arithmetic encoder [18][19] to encode the AC elements of $\mathbf{f}_q$ by assuming an underlying zero mean Laplacian probability distribution. Note the variance of each element of $\mathbf{f}_q$ is different – it depends on the number of non-empty voxels N in each block, the connectivity between the voxels, etc. Fortunately, the eigenvalue decomposition process for finding the transform matrix $\mathbf{\Phi}$ gives us the clue: the eigenvalue of each eigenvector is inversely proportional to the expected variance of the corresponding coefficient.

Thus, instead of computing a probability table for the coefficients, we assume all (unquantized) coefficients follow a continuous scaled Laplacian distribution, with a diversity parameter inversely proportional to the square root of the corresponding eigenvalue. More specifically, we assume the coefficients have the following probability density function:

$$p(x_i) = \frac{\sqrt{\lambda_i}}{2\sigma} \exp\{-\frac{\sqrt{\lambda_i}|x_i|}{\sigma}\}, \tag{8}$$

where x_i is the i^{th} AC coefficient, and λ_i is its corresponding eigenvalue; σ is the base diversity parameter of the underlying Laplacian distribution, which shall be updated every time a new coefficient is encoded.

The probability tables required by the arithmetic encoder are derived from the above Laplacian distribution. After encoding a new coefficient, we update the base diversity parameter as:

$$\sigma \leftarrow \sqrt{\left(k\sigma^2 + \frac{1}{2}\lambda_i \|x_{iq}\|^2\right)/(k+1)}, \tag{9}$$

where k is the total number of previously encoded AC coefficients; x_{iq} is the quantized coefficient of x_i. The parameter σ is initialized to a small constant number at the beginning of each point cloud.

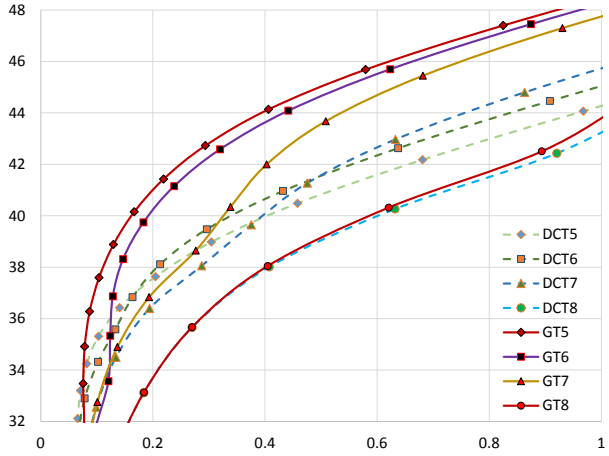

Fig. 2. Compression performance (in dB) vs. bitrate (in bpv) for the proposed encoder using a Graph Transform and DCT for octree level 5, 6, 7 and 8 (corresponding to cube sizes of 16, 8, 4 and 2). The curves are the average of all 6 scenes used in our experiment.

3.2. Encoding the DC term

The DC coefficients need to be handled differently from the AC components. Given a DC coefficient d_i, we first remove the mean by computing $d_i^* = d_i - \sqrt{N_i/N_{i-1}}\hat{d}_{i-1}$, where N_i is the number of connected voxels corresponding to d_i, and $\hat{d}_{i-1}$ is the (decoded) value of last DC term. Afterwards, we assume the difference signal follows a Laplacian distribution:

$$p(d_i^*) = \frac{1}{2\rho\sqrt{N_i}} \exp\{-\frac{|d_i^*|}{\rho\sqrt{N_i}}\}, \tag{10}$$

where ρ is an adaptive diversity parameter. After encoding d_i^*, we update ρ as:

$$\rho \leftarrow \sqrt{\left(k\rho^2 + \frac{\|\hat{d}_i^*\|^2}{2N_i}\right)/(k+1)}, \tag{11}$$

where $\hat{d}_i^*$ is the decoded prediction difference; k is the total number of previously encoded DC coefficients; and ρ is re-initialized to a small constant for each point cloud.

4. EXPERIMENTAL RESULTS

We conducted experiments on 3D point clouds constructed by a real-time high resolution sparse voxelization algorithm as presented in [20]. Our system consists of 8 cameras mounted on a $5 \times 5 \times 5$ foot cube pointing slightly downward on the top edges in an octagonal configuration. A human subject was seated at the center of the rig. The captured videos are processed on a high-end graphics card using a silhouette-based voxel carving algorithm in real-time. See [20] for more details. We applied the proposed algorithm on a total of 6 different scenes, with 2 of them shown in Fig. 3.

The sparse voxelization algorithm outputs an octree directly due to the multi-resolution nature of its design. For our experiments, we limit the octree to 9 levels, which corresponds to a resolution of $512 \times 512 \times 512$ voxels, or roughly 2mm accuracy in the real-world. A typical point cloud for a human upper body contains between

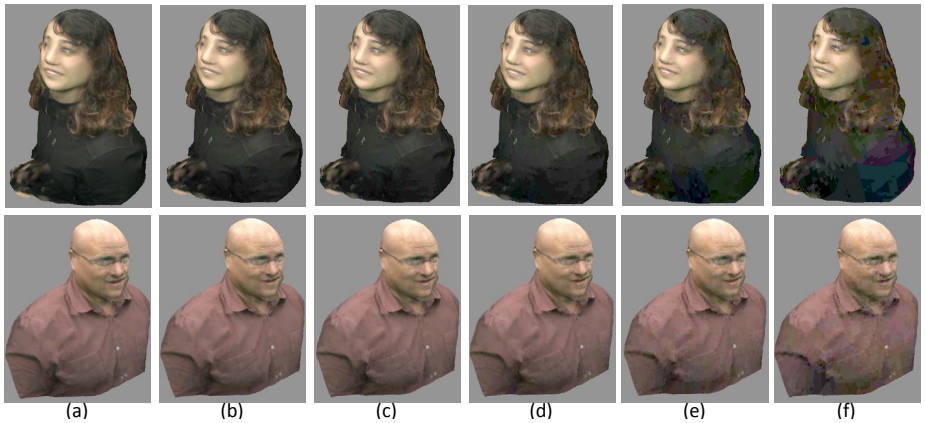

Fig. 3. Rendering results for point clouds compressed with different quantization steps. (a) original point cloud. (b) $Q = 4$. (c) $Q = 8$. (d) $Q = 16$. (e) $Q = 32$. (f) $Q = 64$. Here Q is the quantization step size. All results are using graph transform at level 6 ($8 \times 8 \times 8$ voxels).

500,000–750,000 3D points with color attributes. The octree structure is compressed with the children pattern sequence [21], which we will improve later with more recent schemes such as [5][7]. The experimental results presented in this section focus only on the compression of the color attributes of the generated point cloud.

We had some difficulty in finding an appropriate baseline for comparison. One of the most widely used point cloud encoders is the one built into the Point Cloud Library (PCL) [22]. Comparison between the PCL and our encoder is shown in Table 1, where the reported numbers are the average value of all 6 scenes. It can be seen that the performance of the PCL encoder on color attributes is poor. For example, for a luminance reconstruction error around 38dB, PCL requires over 8.4 bits per voxel (bpv), while our encoder requires less than 0.36 bpv. In other words, the PCL coded bitstream is more than 23 times larger than that coded by the proposed method.

Since one of the objectives of this paper is to introduce the graph transform to the application of point cloud attribute compression, we decide to compare the results against a similar encoder as ours, except it uses the N-point 1D DCT instead of the graph transform $\mathbf{\Phi}$. We point out that in [15] the authors have shown that 1D DCT is a special case of graph transform, where the graph is constructed as a 1D chain and the precision matrix is defined as the standard Laplacian matrix. We would like to show that by allowing voxels to connect to 3D neighbors instead of 1D neighbors (in a depth-first scan order), the resultant graph transform is more effective in decorrelating the signal, thus leading to better compression performance.

The experimental results are shown in Fig. 2. We tested different block sizes, where level 5 block has $16 \times 16 \times 16$ voxels, level 6 block has $8 \times 8 \times 8$ voxels, level 7 block has $4 \times 4 \times 4$ voxels, and level 8 block has $2 \times 2 \times 2$ voxels. It can be seen that graph transform significantly outperforms the N-point 1D DCT for level 5, 6 and 7. At the same bit rate, the PSNR difference can be up to 1-3 dB depending on the operating bit rate. At the same PSNR, graph transform may spend only half of the bit rate compared with DCT. When the block size is too small (e.g., level 8), it is expected that graph transform and DCT will have similar performance, since voxels in the same small block will probably always have very similar colors.

Fig. 2 also shows that for graph transform, the larger the block size, the better the compression performance, since more voxels are decorrelated together in each block. On the other hand, larger block size also implies slower encoding speed. Graph transform is structure dependent, and it cannot be pre-computed due to the large number of variations in point cloud structure. The eigenvalue decomposition of the precision matrix $\mathbf{Q}$ has complexity on the order of $O(N^3)$, therefore the more valid voxels in a block, the slower it runs. For example, on a single core 3.0GHz CPU, given a point cloud with about 520,000 points, level 8 graph transform takes about 0.85 seconds, level 7 takes about 8.2 seconds, level 6 takes 141 seconds, and level 5 takes 109.8 minutes. The trend is very clear. We note that our code is not heavily optimized, and we expect things to run much faster if the graph transforms of different blocks are run in parallel, possibly on a GPU.

In the case of N-point 1D DCT, we note that larger block sizes do not improve coding efficiency. We attribute this phenomenon to possible color discontinuity when forming the 1D voxel chain. For a particular scene, one may be able to find better octree traversal algorithms than the currently adopted depth-first traversal scheme. However, we expect graph transform to always perform better than 1D DCT, since correlations in the 3D space are always better considered in the graph transform.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a novel graph transform based approach for encoding 3D point cloud attributes. Considering that the 3D point cloud's position information can be encoded prior to the attributes, graph transform appears to be an ideal solution since no overhead is incurred to construct the graph for unstructured point clusters. Our experimental results showed a significant improvement over traditional methods.

There are a few directions that can be explored in the future. First, the scheme to construct the graph in each block (Section 2.2) is rather simple. One possible drawback is that it would create too many isolated sub-graphs if the point cloud is not very dense. More sophisticated approaches such as those based on k-nearest neighbors could be used instead. Second, the work in this paper focuses on encoding single point clouds. For sequences of point clouds, such as that generated by our real-time sparse voxelization scheme [20], we need to further explore the temporal relationship between point clouds in neighboring frames.

6. REFERENCES

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

[2] Szymon Rusinkiewicz and Marc Levoy, "Qsplat: A multiresolution point rendering system for large meshes," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 2000, pp. 343–352.

[3] Tilo Ochotta and Dietmar Saupe, "Compression of point-based 3d models by shape-adaptive wavelet coding of multi-height fields," in *Proceedings of the First Eurographics conference on Point-Based Graphics*. Eurographics Association, 2004, pp. 103–112.

[4] Ruwen Schnabel and Reinhard Klein, "Octree-based point-cloud compression," in *SPBG*, 2006, pp. 111–120.

[5] Yan Huang, Jingliang Peng, C-CJ Kuo, and M Gopi, "A generic scheme for progressive point cloud coding," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 14, no. 2, pp. 440–453, 2008.

[6] Wenfei Jiang, Jiang Tian, Kangying Cai, Fan Zhang, and Tao Luo, "Tangent-plane-continuity maximization based 3d point compression," in *Image Processing (ICIP), 2012 19th IEEE International Conference on*. IEEE, 2012, pp. 1277–1280.

[7] Julius Kammerl, Nico Blodow, Radu Bogdan Rusu, Suat Gedikli, Michael Beetz, and Eckehard Steinbach, "Real-time compression of point cloud streams," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 778–785.

[8] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *Signal Processing Magazine, IEEE*, vol. 30, no. 3, pp. 83–98, 2013.

[9] Aliaksei Sandryhaila and José M. F. Moura, "Discrete signal processing on graphs," *IEEE Trans. on Signal Processing*, vol. 61, no. 7, pp. 1644–1656, 2013.

[10] Zachi Karni and Craig Gotsman, "Spectral compression of mesh geometry," in *SIGGRAPH*, 2000.

[11] Yu Gao, Gene Cheung, Thomas Maugey, Pascal Frossard, and Jie Liang, "3d geometry representation using multiview coding of image tiles," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2014.

[12] G. Shen, W.-S. Kim, S. K. Narang, A. Ortega, J. Lee, and H. Wey, "Edge-adaptive transforms for efficient depth map coding," in *PCS*, 2010.

[13] Gene Cheung, Woo-Shik Kim, Antonio Ortega, Junichi Ishida, and Akira Kubota, "Depth map coding using graph based transform and transform domain sparsification," in *Multimedia Signal Processing (MMSP), 2011 IEEE 13th International Workshop on*. IEEE, 2011, pp. 1–6.

[14] Wei Hu, Gene Cheung, Xin Li, and Oscar Au, "Depth map compression using multi-resolution graph-based transform for depth-image-based rendering," in *Image Processing (ICIP), 2012 19th IEEE International Conference on*. IEEE, 2012, pp. 1297–1300.

[15] Cha Zhang and Dinei Florêncio, "Analyzing the optimality of predictive transform coding using graph-based models," *IEEE Signal Processing Letters*, vol. 20, no. 1, pp. 106–109, 2013.

[16] Sunil K Narang, Yung-Hsuan Chao, and Antonio Ortega, "Critically sampled graph-based wavelet transforms for image coding," in *Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2013 Asia-Pacific*. IEEE, 2013, pp. 1–4.

[17] Havard Rue and Leonhard Held, *Gaussian Markov Random Fields: Theory and Applications*, Chapman and Hall/CRC, 2005.

[18] Nobutaka Kuroki, Takahiro Manabe, and Masahiro Numa, "Adaptive arithmetic coding for image prediction errors," in *ISCAS*, 2004.

[19] Paul G Howard and Jeffrey Scott Vitter, *Practical implementations of arithmetic coding*, Springer, 1992.

[20] Charles Loop, Cha Zhang, and Zhengyou Zhang, "Real-time high-resolution sparse voxelization with application to image-based modeling," in *High Performance Graphics*, 2013.

[21] Jyrki Katajainen and Erkki Mäkinen, "Tree compression and optimization with applications," *Int. J. Found. Comput. Sci.*, vol. 1, no. 4, pp. 425–448, 1990.

[22] "The point cloud library," in *http://www.pointclouds.org/*.