Point Cloud Attribute Compression with Graph Transform

Lu Yu

iMorpheus.ai

August 11th 2017

Background

We discussed 3D point cloud compression with octree last week. In that case, each point has three coordinates but no attribute, such as color or normal vector. Today we introduce the following paper that deals with compression of these attributes. The main approach is graph transform on a weighted undirected graph via Laplacian matrix.

Zhang, Cha, Dinei Florencio, and Charles Loop. **Point cloud attribute compression with graph transform.** *Image Processing (ICIP), 2014 IEEE International Conference on.* IEEE, 2014.

Data structure and size

Point index	3D coordinates	Color in YUV space
Data type	32 bit float	16 bit integer
Size	$4 \times 3 = 12$ bytes	$2 \times 3 = 6$ bytes

We consider static point cloud. Color in YUV space is used as an example of attribute.

We assume that coordinate information has already been processed (both encoding and decoding) via octree method discussed last week. The coordinates can be compressed to about 1.34 bytes per point with accuracy 1 mm.

Graph construction

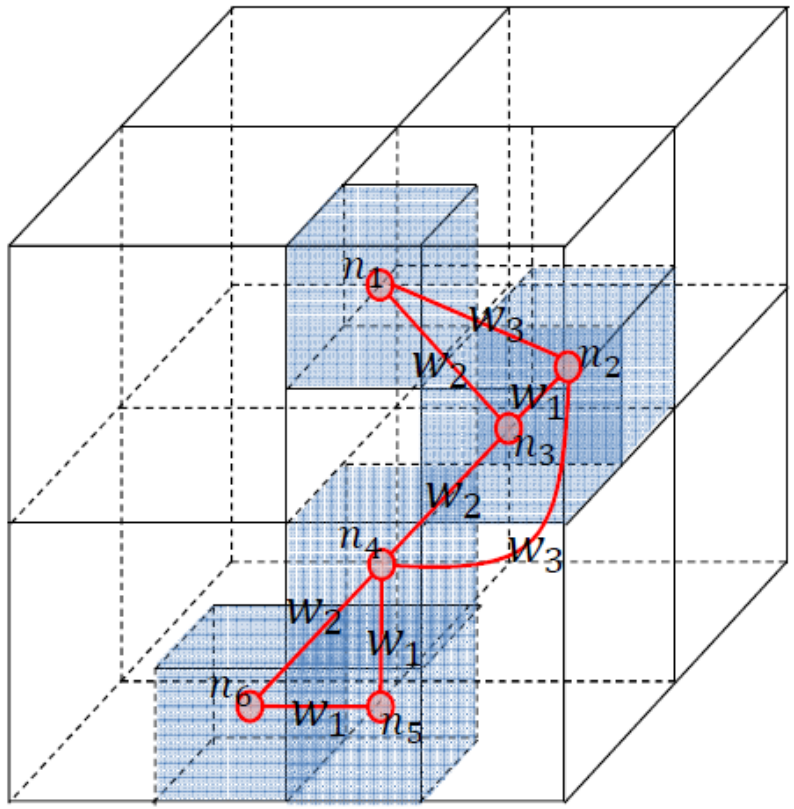

Fig. 1. Forming a graph based on point clouds on an octree block. Here $n_1, \cdots, n_6$ are the nodes of the graph, and $w_1, \cdots, w_3$ are weight values defined on the graph edges.

Graph construction

1. Choose a small chunk of octree (e.g. $4 \times 4 \times 4$ voxels) and consider all points in side it.
2. A point is considered adjacent to 26 points in its neighboring $3 \times 3 \times 3$ voxels.
3. Edge weight is inversely proportional to distance. $w_1 = 1, w_2 = \frac{1}{\sqrt{2}}, w_3 = \frac{1}{\sqrt{3}}$
4. Construct adjacency matrix $\mathbf{A}$, degree matrix $\mathbf{D}$ and Laplacian matrix $\mathbf{L}$.

Graph Laplacian matrix

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

$$\mathbf{D} = \text{diag}(d_1, \ldots, d_6) \text{ with } d_i = \sum_j w_{ij}$$

$$\mathbf{L} = \mathbf{D} - \mathbf{A}$$

Note that $\mathbf{L}$ is symmetric diagonally dominated and therefor positive semi-definite.

Signal transform

Perform eigenvalue decomposition on $\mathbf{L} = \mathbf{\Phi}\mathbf{\Lambda}\mathbf{\Phi}^{-1}$. Let $\mathbf{y}$ be the column vector that represents Y (brightness) value of all points in the selected chunk. It is transformed into $\mathbf{f} = \mathbf{\Phi}\mathbf{y}$ and quantized as $\mathbf{f}_q = \text{round}(\mathbf{f}/Q)$, where Q is quantization step size. U and V values can be processed similarly.

The number of zeros in the diagonal of $\mathbf{\Lambda}$ is the number of connected components of the graph. Their corresponding components of $\mathbf{f}_q$ are called DC terms. They correspond to the average of connected components, multiplied by the square root of the number of points in each component.

The other components of $\mathbf{f}_q$ are called AC terms. DC and AC terms have different statistic properties and are thus encoded differently.

AC terms

We use arithmetic encoding for AC terms. They are assumed to follow Laplacian distribution. For each AC coefficient x_i in $\mathbf{f}_q$,

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

where λ_i is the corresponding eigenvalue and σ is the underlying Laplacian distribution. σ is initialized to a small number and updated when a new coefficient is encoded as follows

$$\sigma = \sqrt{(k\sigma^2 + \frac{1}{2}\lambda_i|x_{iq}|^2)/(k+1)},$$

where k is the total number of previously encoded AC coefficients and $|x_{iq}|$ is the quantized coefficient of x_i.

DC terms

Given a DC coefficient d_i, we first remove the mean by $d_i^* = d_i - \sqrt{N_i/N_{i-1}}\hat{d}_{i-1}$, where N_i is the number of connected points corresponding to d_i and $\hat{d}_{i-1}$ is the decoded value of last DC term. We now assume the difference signal follows Laplacian distribution

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

where ρ is an adaptive diversity parameter and initialized to a small number. After encoding d_i^*, ρ is updated as

$$\rho = \sqrt{(k\rho^2 + \frac{|\hat{d}_i^*|^2}{2N_i})/(k+1)},$$

where d_i^* is the decoded prediction difference and k is the total number of previously encoded DC coefficients.

Summary of encoding procedure

1. Build octree structure for point cloud and encode coordinates.
2. For $k \times k \times k$ voxel chunk of octree, construct a weighted undirected graph for points in it.
3. Calculate graph Laplacian matrix and perform eigenvalue decomposition.
4. Transform and quantize attribute signals.
5. Encode AC components.
6. Encode DC components.

Results

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.

Results

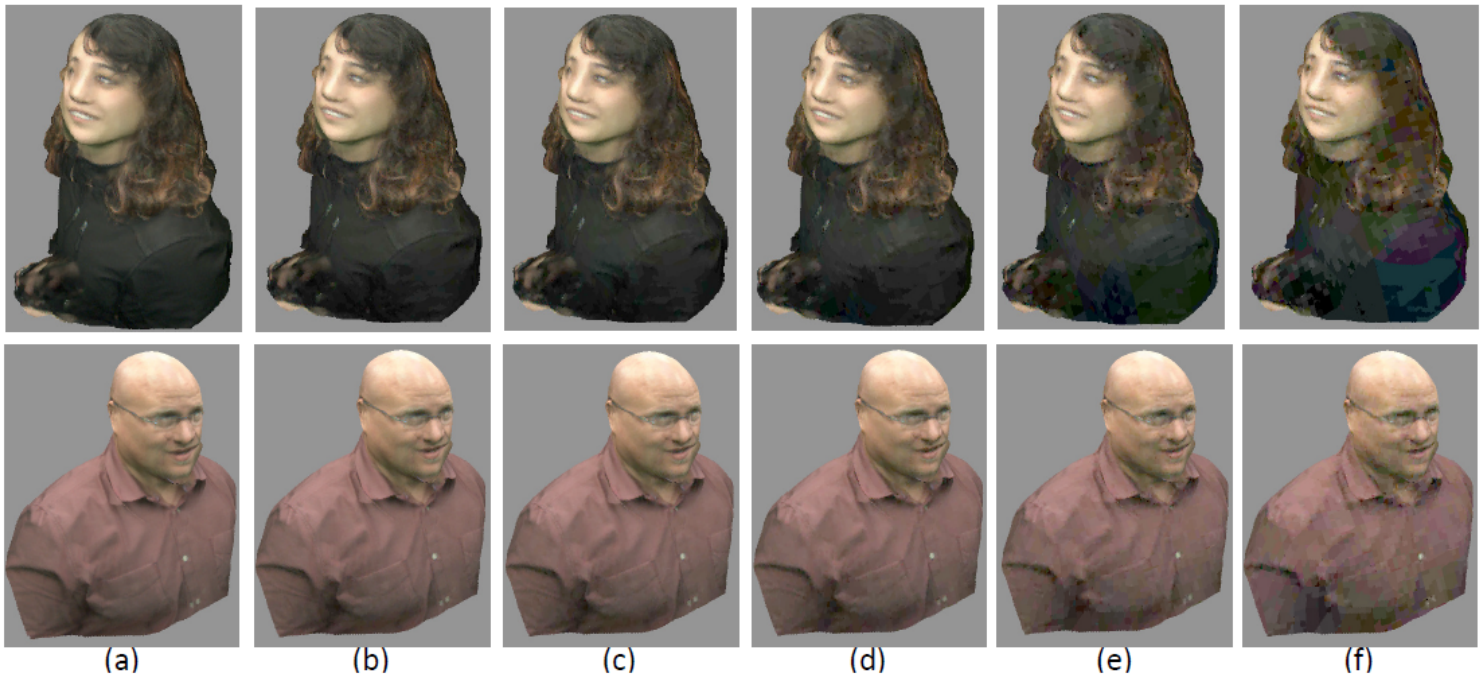

Fig. 3. Rendering results for point clouds compressed with different quantization steps. (a) original point cloud (b) $Q = 4$ (c) $Q = 8$ (d) $Q = 16$ (e) $Q = 32$ (f) $Q = 64$. Here Q is the quantization step size. All results are using graph transform on $8 \times 8 \times 8$ voxel chunk.