LEARNING CONVOLUTIONAL TRANSFORMS FOR LOSSY POINT CLOUD GEOMETRY COMPRESSION

Maurice Quach  Giuseppe Valenzise  Frederic Dufaux

L2S, CNRS, CentraleSupélec, Université Paris-Saclay

ABSTRACT

Efficient point cloud compression is fundamental to enable the deployment of virtual and mixed reality applications, since the number of points to code can range in the order of millions. In this paper, we present a novel data-driven geometry compression method for static point clouds based on learned convolutional transforms and uniform quantization. We perform joint optimization of both rate and distortion using a trade-off parameter. In addition, we cast the decoding process as a binary classification of the point cloud occupancy map. Our method outperforms the MPEG reference solution in terms of rate-distortion on the Microsoft Vox-lized Upper Bodies dataset with 51.5% BDBR savings on average. Moreover, while octree-based methods face exponential diminution of the number of points at low bitrates, our method still produces high resolution outputs even at low bitrates.

Index Terms— point cloud geometry compression, convolutional neural network, rate-distortion optimization

1. INTRODUCTION

Point clouds are an essential data structure for Virtual Reality (VR) and Mixed Reality (MR) applications. A point cloud is a set of points in the 3D space represented by coordinates \( x, y, z \) and optional attributes (for example color, normals, etc.). Point cloud data is often very large as point clouds easily range in the millions of points and can have complex sets of attributes. Therefore, efficient point cloud compression (PCC) is particularly important to enable practical usage in VR and MR applications.

The Moving Picture Experts Group (MPEG) is currently working on PCC. In 2017, MPEG issued a call for proposals (CfP) and in order to provide a baseline, a point cloud codec for tele-immersive video [1] was chosen as the MPEG anchor. To compare the proposed compression solutions, quality evaluation metrics were developed leading to the selection of the point-to-point (D1) and point-to-plane (D2) as baseline metrics [2]. The point to point metric, also called D1 metric, is computed using the Mean Squared Error (MSE) between the reconstructed points and the nearest neighbors in the reference point cloud. The point-to-plane metric, also called D2 metric, uses the surface plane instead of the nearest neighbor.

Research on PCC can be categorized along two dimensions. On one hand, one can either compress point cloud geometry, i.e., the spatial position of the points, or their associated attributes. On the other hand, we can also separate works focusing on compression of dynamic point clouds, which contain temporal information, and static point clouds.

In this work, we focus on the lossy compression of static point cloud geometry. In PCC, a precise reconstruction of geometric information is of paramount importance to enable high-quality rendering and interactive applications. For this reasons, lossless geometry coding has been investigated recently in MPEG, but even state-of-the-art techniques struggle to compress beyond about 2 bits per occupied voxels (bpov) [3]. This results in large storage and transmission costs for rich point clouds. Lossy compression proposed in the literature, on the other hand, are based on octrees which achieve variable-rate geometry compression by changing the octree depth. Unfortunately, lowering the depth reduces the number of points exponentially. As a result, octree based lossy compression tends to produce “blocky” results at the rendering stage with medium to low bitrates. In order to partially attenuate this issue, [4] proposes to use wavelet transforms and volumetric functions to compact the energy of the point cloud signal. However, since they still employ an octree representation, their method exhibits rapid geometry degradation at lower bitrates. While previous approaches use hand-crafted transforms, we propose here a data driven approach based on learned convolutional transforms which directly works on voxels.

Specifically, we present a method for learning analysis and synthesis transforms suitable for point cloud geometry compression. In addition, by interpreting the point cloud geometry as a binary signal defined over the voxel grid, we cast decoding as the problem of classifying whether a given voxel is occupied or not. We train our model on the ModelNet40 mesh dataset [5, 6], test its performance on the Microsoft Voxelized Upper Bodies (MVUB) dataset [7] and compare it with the MPEG anchor [1]. We find that our method outperforms the anchor on all sequences at all bitrates. Additionally, in contrast to octree-based methods, ours does not exhibit exponential diminution in the number of points when lowering the bitrate. We also show that our model generalizes well by
3. PROPOSED METHOD

In this section, we describe the proposed method in more details.

3.1. Definitions

First, we define the set of possible points at resolution \( r \) as \( \Omega_r = [0..r]^3 \). Then, we define a point cloud as a set of points \( S \subseteq \Omega_r \) and its corresponding voxel grid \( v_S \) as follows:

\[
    v_S : \Omega_r \rightarrow \{0, 1\},
    z \mapsto \begin{cases} 
        1, & \text{if } z \in S \\
        0, & \text{otherwise}.
    \end{cases}
\]

For notational convenience, we use \( s^3 \) instead of \( s \times s \times s \) for filter sizes and strides.

3.2. Model

We use a 3D convolutional auto-encoder composed of an analysis transform \( f_a \), followed by a uniform quantizer and a synthesis transform \( f_s \).

Let \( x = v_S \) be the original point cloud. The corresponding latent representation is \( y = f_a(x) \). To quantize \( y \), we introduce a quantization function \( Q \) so that \( \hat{y} = Q(y) \). This allows us to express the decompressed point cloud as \( \hat{x} = \hat{v}_S = f_s(\hat{y}) \). Finally, we obtain the decompressed point cloud \( \hat{x} = \hat{v}_S = \text{round}(\min(0, \max(1, \hat{x}))) \) using element-wise minimum, maximum and rounding functions.

In our model, we use convolutions and transpose convolutions with same padding and strides. They are illustrated in Figure 2 and defined as follows:

- Same (half) padding pads the input with zeros so that the output size is equal to the input size.
- Convolution performed with unit stride means that the convolution filter is computed for each element of the input array. When iterating the input array, strides specify the step for each axis.
- Convolution can be seen as matrix multiplication and transpose convolution can be derived from this. In particular, we can build a sparse matrix \( C \) with non-zero elements corresponding to the weights. The transpose convolution, also called deconvolution, is obtained using the matrix \( C^T \) as a layout for the weights.

Using these convolutional operations as a basis, we learn analysis and synthesis transforms structured as in Figure 1 using the Adam optimizer [17] which is based on adaptive estimates of first and second moments of the gradient.

We handle quantization similarly to [11]. \( Q \) represents element-wise integer rounding during evaluation and \( Q \) adds uniform noise between \(-0.5 \) and \( 0.5 \) to each element during
training which allows for differentiability. To compress \( Q(y) \), we perform range coding and use the Deflate algorithm, a combination of LZ77 and Huffman coding [18] with shape information on \( x \) and \( y \) added before compression. Note however that our method does not assume any specific entropy coding algorithm.

Our decoding process can also be interpreted as a binary classification problem where each point \( z \in \Omega_r \) of the voxel grid is either present or not. This allows us to decompose \( \hat{x} = \hat{v}_S \) into its individuals voxels \( z \) whose associated value is \( p_z \). However, as point clouds are usually very sparse, most \( v_s(z) \) values are equal to zero. To compensate for the imbalance between empty and occupied voxels we use the \( \alpha \)-balanced focal loss as defined in [19]:

\[
FL(p_z^t) = -\alpha_z(1 - p_z^t)^\gamma \log(p_z^t)
\]

with \( p_z^t \) defined as \( p_z \) if \( v_s(z) = 1 \) and \( 1 - p_z \) otherwise. Analogously, \( \alpha_z \) is defined as \( \alpha \) when \( v_s(z) = 1 \) and \( 1 - \alpha \) otherwise. The focal loss for the decompressed point cloud can then be computed as follows:

\[
FL(\hat{x}) = \sum_{z \in S} FL(p_z^t).
\]

Our final loss is \( L = \lambda D + R \) where \( D \) is the distortion calculated using the focal loss and \( R \) is the rate in number of bits per input occupied voxel (bpov). The rate is computed differently during training and during evaluation. On one hand, during evaluation, as the data is quantized, we compute the rate using the number of bits of the final compressed representation. On the other hand, during training, we add uniform noise in place of discretization to allow for differentiation. It follows that the probability distribution of the latent space \( Q(y) \) during training is a continuous relaxation of the probability distribution of \( Q(y) \) during evaluation which is discrete. As a result, entropies computed during training are actually differential entropies, or continuous entropies, while entropies computed during evaluation are discrete entropies. During training, we use differential entropy as an approximation of discrete entropy. This makes the loss differentiable which is primordial for training neural networks.

4. EXPERIMENTAL RESULTS

We use train, evaluation and test split across two datasets. We train and evaluate our network on the ModelNet40 aligned dataset [5, 6]. Then, we perform tests on the MVUB dataset and we compare our method with the MPEG anchor [1].

We perform our experiments using Python 3.6 and TensorFlow 0.12. We use \( N = 32 \) filters, a batch size of 64 and Adam with \( lr = 10^{-4}, \beta_1 = 0.9 \) and \( \beta_2 = 0.999 \). For the focal loss, we use \( \alpha = 0.9 \) and \( \gamma = 2.0 \).

To compute distortion, we use the point-to-plane symmetric PSNR computed with the \texttt{pc_error} MPEG tool [20].

4.1. Datasets

The ModelNet40 dataset contains 12,311 mesh models from 40 categories. This dataset provides us with both variety and quantity to ensure good generalization when training our network. To convert this dataset to a point cloud dataset, we first perform sampling on the surface of each mesh. Then, we translate and scale it into a voxel grid of resolution \( r \). We use this dataset for training with a resolution \( r = 64 \).

The MVUB dataset [7] contains 5 sequences captured at 30 fps during 7 to 10 seconds each with a total of 1202 frames. We test our method on each individual frame with a resolution \( r = 512 \). In other words, we evaluate performance for intra-frame compression on each sequence.

We compute RD curves for each sequence of the test dataset. For our method, we use the following \( \lambda \) values to compute RD points: \( 10^{-4}, 5 \times 10^{-5}, 10^{-5}, 5 \times 10^{-6} \) and \( 10^{-6} \). For each sequence, we average distortions and bitrates over time for each \( \lambda \) to obtain RD points. For the MPEG anchor, we use the same process with different octree depths.

To evaluate distortion, we use the point-to-plane symmetric PSNR [20] \( e_{symm}(A,B) = \min(e(A,B),e(B,A)) \) where \( e(A,B) \) provides the point-to-plane PSNR between points in \( A \) and their nearest neighbors in \( B \). This choice is due to the fact that original and reconstructed point clouds may have a very different number of points, e.g., in octree-based methods the compressed point cloud has significantly less points than the original, while in our method it is the opposite. In the rest of this section, we refer to the point-to-plane symmetric PSNR as simply PSNR.

Our method outperforms the MPEG anchor on all sequences at all bitrates. The latter has a mean bitrate of 0.719 bpov and a mean PSNR of 16.68 dB while our method has a mean bitrate of 0.691 and a mean PSNR of 24.11 dB. RD curves and the Bjontegaard-delta bitrates (BDBR) for each...
sequence are reported in Figure 3. Our method achieves 51.5% BDBR savings on average compared to the anchor.

In Figure 4, we show examples on the first frame of the Phil sequence. Our method achieves lower distortion at similar bitrates and produces more points than the anchor which increases quality at low bitrates while avoiding “blocking” effects. This particular example shows that our method produces 218 times more points than the anchor at similar bitrates. In other words, both methods introduce different types of distortions. Indeed, the number of points produced by octree structures diminishes exponentially when reducing the octree depth. Conversely, our method produces more points at lower bitrates as the focal loss penalizes false negatives more heavily.

5. CONCLUSION

We present a novel data-driven point cloud geometry compression method using learned convolutional transforms and a uniform quantizer. Our method outperforms the MPEG Anchor on the MVUB dataset in terms of rate-distortion with 51.5% BDBR savings on average. Additionally, in contrast to octree-based methods, our model does not exhibit exponential diminution in the number of output points at lower bitrates. This work can be extended to the compression of attributes and dynamic point clouds.

6. ACKNOWLEDGMENTS

This work was funded by the ANR ReVeRy national fund (REVERY ANR-17-CE23-0020).
7. REFERENCES


