IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received June 25, 2019, accepted August 2, 2019, date of publication August 14, 2019, date of current version August 28, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2935253

Real-Time Streaming Point Cloud Compression for 3D LiDAR Sensor Using U-Net

CHENXI TU[1], EIJIRO TAKEUCHI[1], (Member, IEEE), ALEXANDER CARBALLO[2], (Member, IEEE), AND KAZUYA TAKEDA[2], (Senior Member, IEEE)

[1]Department of Intelligent System, Graduate School of Informatics, Nagoya University, Nagoya 464-8603, Japan
[2]Institutes of Innovation for Future Society, Nagoya University, Nagoya 464-8603, Japan

Corresponding author: Chenxi Tu (tu.chenxi@g.sp.m.is.nagoya-u.ac.jp)

This work was supported by the Core Research for Evolutional Science and Technology (CREST), Japan Science and Technology Agency (JST).

ABSTRACT Point cloud data from LiDAR sensors is the currently the basis of most L4 autonomous driving systems. Sharing and storing point clouds will also be important for future applications, such as accident investigation or V2V/V2X networks. Due to the huge volume of data involved, storing point clouds collected over long periods of time and transmitting point clouds in real-time are difficult tasks, making compression an indispensable step before storing or transmitting. Previous streaming point cloud compression methods, such as octree compression or video compression-based approaches, have difficulty compressing this data in real-time into very small volumes with low information loss. To reduce temporal redundancy efficiently and rapidly, in this paper we propose a real-time streaming point cloud data compression method using U-net. By utilizing raw packet data from LiDAR sensors, we can store 3D point cloud information losslessly in a 2D matrix, and convert streaming point cloud data into a video-like format. By designating some frames as reference frames and then using U-net to interpolate the remaining LiDAR frames, we can greatly reduce temporal redundancy. Our use of U-net was inspired by a video interpolation approach employed in another study. Noise in LiDAR data is a big issue which significantly affects network training and compression results. In this paper, we propose a padding strategy to alleviate the negative impact of this noise. As a result of these improvements, our proposed method can outperform octree compression, MPEG-based compression and our previously proposed SLAM-based compression method.

INDEX TERMS Point cloud, data compression, deep learning, U-net.

I. INTRODUCTION

A point cloud is a collection of points which can be thought of as a sampling of the real-world. Using a point cloud, we can determine the exact positions of obstacles and better understand the surrounding environment. Ever since point clouds detected with 3D LiDAR sensors demonstrated the usefulness of this technology at the DARPA urban challenge in 2007, this has been widely used by level 4 autonomous driving systems [2]. Shared and stored streaming point cloud data is also likely be an important component of accident investigation and future V2V (vehicle-to-vehicle) and V2X (vehicle-to-everything) systems. Streaming point cloud data is a type of "big data" however, and one hour of point cloud data from the Velodyne HDL-64 sensor mentioned above can result in over 100GB of data, which is too large to realistically share or store using currently available technology. Thus, developing methods of compressing this data has become an indispensable task.

In order to compress a streaming point cloud, we need to deal with spatial and temporal redundancy. However, it's difficult to directly reduce a point cloud's spatial redundancy because of its sparsity and disorder. Most approaches choose to format point cloud data into a tree structure or 2D matrix. In our previous work [3], we demonstrated that formatting raw LiDAR packet data in a 2D matrix, and then compressing it using an existing image compression method, is an efficient strategy for reducing a point cloud's spatial redundancy.

To reduce the temporal redundancy of streaming data, a portion of the frames are designated as reference/key frames and these frames are then used to predict the content of the enclosed frames. One example of this approach is MPEG compression, which uses motion compensation to predict video frames. In contrast to video data, the pixels

The associate editor coordinating the review of this article and approving it for publication was Ming Luo.

in adjacent frames of 2D formatted LiDAR data not only translate, their values (usually representing distance) can also change. In addition, as a low-resolution sampling of the real-world, some pixels (points) may not appear in the following frame. And adjacent pixels do not always have similar motion because the laser modules in a LiDAR apparatus are usually not distributed in a line or at a single point, and have varying orientations. All of these factors make it difficult to estimate a pixel's motion and predict the content of enclosed frames using video compression strategies.

However, since streaming point cloud data is visually continuous, we believe pixel motion between a pair of 2D formatted LiDAR data frames could be estimated and used to predict the content of the enclosed frames, but a new approach is needed. Deep learning, which is famous for its ability to learn features and patterns from data, could be a solution.

Inspired by deep learning-based video interpolation approaches [1], we propose using U-net to predict the enclosed frames between each pair of reference/key frames to reduce the temporal redundancy of streaming point cloud data. Meanwhile, spatial redundancy is reduced by formatting each frame's LiDAR data into a 2D matrix and further coding it.

In comparison with the octree, MPEG-based and SLAM-based compression methods, the compression method proposed in this study can be used in real-time and results in a higher compression rate with less information loss.

The main contributions of this paper are as follows:

- A deep learning-based approach to the compression of streaming point cloud data from a 3D LiDAR sensor, which is, as far as we know, the first use of deep learning for streaming point cloud compression.
- A pre-processing strategy to alleviate the influence of sensor noise.
- Proposed method can be implemented in real-time and outperforming octree compression, MPEG based compression and our previous SLAM based compression methods.

II. RELATED WORK

In order to effectively compress a streaming point cloud, we need to reduce both its spatial and temporal redundancy. Because point clouds are sparse and disorderly, it's difficult to directly decrease their spatial redundancy, therefore point clouds are usually converted into a different format for subsequent processing.

3D modeling has been an important subject in the field of computer graphics research for some time. 3D mesh compression [4], which has been studied for more than 25 years, can be regarded as the forerunner of 3D point cloud compression, and it directly inspired several early point cloud compression methods. Inspired by this breakthrough, various other 3D point cloud compression methods were then developed, many of which were height map-based methods using strategies similar to the one used in 3D mesh compression. Pauly and Gross [5] were the first to propose using height maps to process point cloud data. Other approaches, such as those of Hubo *et al.* [6] and Ochotta and Saupe [7], [8] were then developed that used height mapping for point cloud compression in conjunction with various coding methods. Schnabel *et al.* [9], [10] suggested using geometric primitives such as planes, spheres, cylinders, cones and tori with height maps in order to compress point cloud data. A method of real-time, continuous point cloud data compression based on a height map was then proposed by Golla and Klein [11]. Other methods did not use height maps per se, but also divided point clouds into patches, such as the method proposed by Morell *et al.* [12], which used triangles to represent each plane.

Other approaches involved converting 3D point cloud data into a 2D format, rather than decomposing one frame of the point cloud into multiple images, as with height map-based methods. Houshiar and Nüchter [13] proposed mapping point clouds onto panorama images using equirectangular projection. Kohira and Masuda [14] mapped point cloud data onto 2D pixels using GPS, time and the parameters of the laser scanner. Directly converting 3D point cloud data into a 2D image will inevitably result in the loss of information, therefore some methods have utilized the LiDAR scanner's operating principle and targeted the raw data from the LiDAR sensor, which is known as packet data. Yin *et al.* [15] focused on this raw packet data and on the LiDAR system's data format. In a previous study [3], we proposed converting LiDAR packet data into a 2D matrix and then using an existing image compression method to compress the data. Except for image compression method, leaning based method has demonstrated its ability in compression 2D data, even outperform existing methods [16]. Methods based on the use of a recurrent neural network (RNN) [17] can also be used to compress 2D formatted LiDAR data, especially packet data, which is usually irregular when in a 2D format. Instead of converting point cloud data into a 2D format, Hayes also took into account the operating principle behind LiDAR and proposed converting a point cloud into a form that is susceptible to wavelet transformation [18].

Other compression methods store and compress 3D point cloud data more directly, using tree structures, for example. Gumhold *et al.* [19] used prediction trees, and Hubo *et al.* [20] used kd-trees, while Schnabel and Elseberg [21] and Elseberg *et al.* [22] used octrees, a method which is now widely used in the field of 3D graphics. Kammerl *et al.* [23] also used octrees to represent the spatial locations of each point, and by using a double-tree structure to calculate the difference (exclusive or) between the octrees of adjacent frames, redundancy within a time series was also reduced for streaming. Kammer's method can be used to control information loss quantitatively and allows the real-time compression of streaming point cloud data. Thanou *et al.* [24] proposed a method of streaming point cloud data compression which includes a special motion estimation module; here, octrees are used to divide 3D point clouds into many occupied voxels,

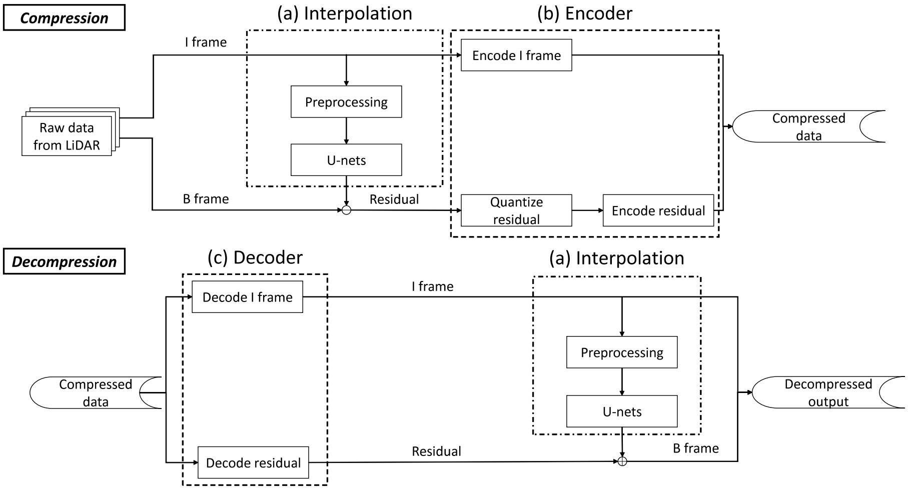


FIGURE 1. **Flowcharts illustrating the proposed method.**

while the points in each voxel (or leaf) are represented by a weighted, undirected graph.

Reducing temporal redundancy is very important for streaming data compression and is highly related to data format. Using a portion of the frames (key/reference frames) to predict the enclosed frames is usually more efficient than only focusing on adjacent frames [23] for reducing temporal redundancy. Since converting packet data into a 2D matrix is a suitable method for reducing spatial redundancy, the key to reducing temporal redundancy for streaming point cloud compression is figuring out how to use the reference frames to predict the remaining enclosed frames.

Using a video compression method is a logical approach for reducing temporal redundancy [3], but this is actually not a very good strategy, as we mentioned in Section I. In [25], we proposed predicting LiDAR frames by simulating LiDAR's operating process, however the high calculation cost involved made this difficult to achieve in real-time.

Point cloud frame prediction is similar to a video frame interpolation task which aims to increase the fps (frames per second) of video data. Most video frame interpolation methods focus on interpolating one frame between each pair of reference frames, and U-net-based approaches have recently achieved the best performance in this area [26], [27]. To interpolate an arbitrary number of enclosed frames, Jiang *et al.* [1] proposed using one additional U-net for refining optical flow to each interpolated frame.

Inspired by the work of Jiang et al., in this paper we use two U-nets to interpolate the frames between each pair of reference frames. In addition, we propose the use of preprocessing to further improve performance. By also using an encoder to reduce spatial redundancy, our proposed method can achieve real-time compression of streaming point cloud data, outperforming previously proposed methods [3], [23], [25].

III. OVERVIEW

As mentioned in Section I, to compress streaming point cloud data efficiently, we need to reduce both spatial and temporal redundancy.

To deal with the sparsity and disorder of point cloud data, before further processing, we losslessly convert each frame's raw LiDAR packet data into a 2D matrix before compression. Please refer to Appendix for details.

To reduce temporal redundancy, in subsequent processing we designate a portion of the frames as reference frames and use them to predict the enclosed frames, which is a widely used strategy for streaming data compression. In the following sections, we will refer to the reference frames as I-frames (intra-coded frames) and to the remaining enclosed frames predicted by the reference frames as B-frames (bi-directionally predicted frames), which is a conventional approach for video compression tasks. The number of B frames between each pair of I frames is fixed by parameter n. For example, if $n = 3$, the input data, in the form of a streaming LiDAR frame sequence, will be formatted as "IBBBIBBBI...".

Fig. 1 shows compression and decompression flowcharts illustrating our proposed method. The whole compression, decompression process can be divided into three parts: (a) interpolation and (b) encoder and (c) decoder.

The (a) Interpolation modules aim at reducing temporal redundancy. Given a pair of I frames, the interpolation module predicts the B frames between them using U-nets. The residual between the predicted B-frames and the true B-frames is then calculated. The interpolation module is the most important module of the process, as the quality of interpolation directly affects the amount of information loss and the storage volume needed after compression. Details are provided in Section IV.

The (b) encoder is used to reduce spatial redundancy between individual frames. In order to retain the information

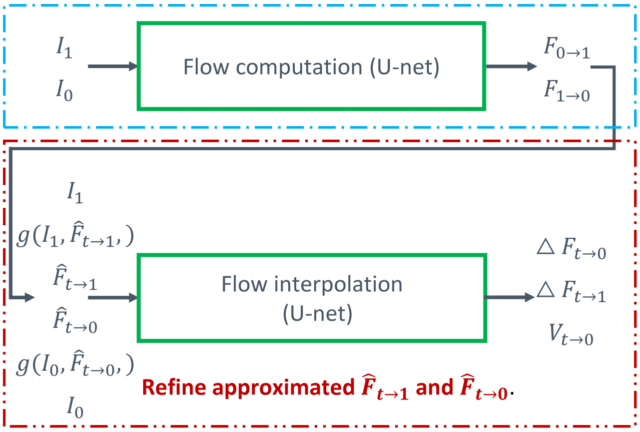


FIGURE 2. Overview of whole frame interpretation network.

contained in the reference frames as accurately as possible, during coding the I-frames are compressed losslessly, while the residual B-frames are quantized first and then coded. By altering the quantization step, we can tune information loss vs. compression rate. Details are provided in Section V.

During decompression, the (c) decoder can be thought of as carrying out an inverse process to that of the encoder.

IV. INTERPOLATION MODULE

A. NETWORK OVERVIEW

Inspired by Jiang *et al.* [1], given two I frames I_0 and I_1, which are 2D matrices of reformatted LiDAR packet data, our proposed method uses two U-nets to infer the enclosed B frames at time t: B_t. Here $t = 1/n, 2/n, \dots (n-1)/n$.

Utilizing optical flow to interpolate 2D frames is a popular and efficient strategy, so our proposed method also follows this approach. To interpolate the frames between I_0 and I_1, optical flow $F_{0\to1}$ (from I_0 to I_1) and $F_{1\to0}$ (from I_1 to I_0) should be calculated first. Since U-net has proven its capability for calculating optical flow in many studies [26], [27], our proposed method also uses U-net to obtain $F_{0\to1}$ and $F_{1\to0}$, as shown in the blue box in Fig 2. We call this U-net a flow computation network.

Basically, given $F_{0\to1}$ and $F_{1\to0}$, we can linearly approximate $\hat{F}_{t\to1}$ and $\hat{F}_{t\to0}$ as follows:

$$\begin{aligned} \hat{F}_{t\to1} &= (1-t)F_{0\to1} \quad \text{or } \hat{F}_{t\to1} = -(1-t)F_{1\to0} \\ \hat{F}_{t\to0} &= -tF_{0\to1} \quad \text{or } \hat{F}_{t\to0} = tF_{1\to0}, \end{aligned} \tag{1}$$

Considering it's better to combine the bi-directional optical flow $F_{0\to1}$ and $F_{1\to0}$ to calculate $F_{t\to1}$ and $F_{t\to0}$, we assume $F_{0\to1} = -F_{1\to0}$, and express equation 1 as follows:

$$\begin{aligned} \hat{F}_{t\to0} &= -(1-t)tF_{0\to1} + t^2F_{1\to0} \\ \hat{F}_{t\to1} &= (1-t)^2F_{0\to1} - t(1-t)F_{1\to0}. \end{aligned} \tag{2}$$

Linearly approximating $F_{t\to1}$ and $F_{t\to0}$ is not sufficient to obtain our desired results however, so following Jiang *et al.* [1] another U-net is used to refine $\hat{F}_{t\to1}$ and $\hat{F}_{t\to0}$ as shown in the red box of Fig 2. We call this U-net a flow interpolation network. The inputs of the interpolation network are $I_0, I_1, \hat{F}_{t\to1}, \hat{F}_{t\to0}, g(I_0, \hat{F}_{t\to0})$ and $g(I_1, \hat{F}_{t\to1})$. Here $g(\cdot,\cdot)$ is a *backward warping* function, which is implemented using bilinear interpolation [28], [29]. In other words, we can approximate B_t using $g(I_0, \hat{F}_{t\to0})$ or $g(I_1, \hat{F}_{t\to1})$.

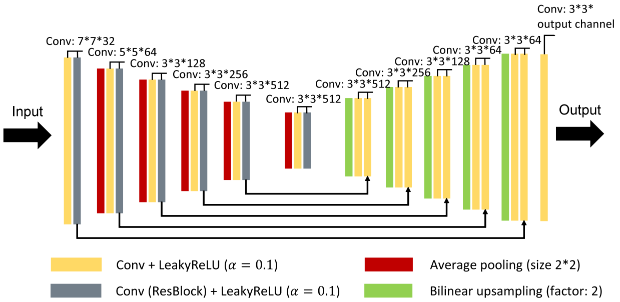

Conv: 7*7*32
Conv: 5*5*64
Conv: 3*3*128
Conv: 3*3*256
Conv: 3*3*512
Conv: 3*3*512
Conv: 3*3*512
Conv: 3*3*256
Conv: 3*3*128
Conv: 3*3*64
Conv: 3*3*64
Conv: 3*3* output channel
Input
Output
Conv + LeakyReLU ($\alpha = 0.1$)
Conv (ResBlock) + LeakyReLU ($\alpha = 0.1$)
Average pooling (size 2*2)
Bilinear upsampling (factor: 2)

FIGURE 3. Detail of U-nets. For all convolution layers, stride = 1 and padding = (filter size + 1)/2.

The outputs of the interpolation network are $\Delta F_{t\to0}$, $\Delta F_{t\to1}$ and $V_{t\leftarrow0}$. $\Delta F_{t\to0}$ and $\Delta F_{t\to1}$ are intermediate optical flow residuals. According to Jiang *et al.* [1], it's better to predict these residuals than to directly output the refined optical flow:

$$\begin{aligned} F_{t\to0} &= \hat{F}_{t\to0} + \Delta F_{t\to0} \\ F_{t\to1} &= \hat{F}_{t\to1} + \Delta F_{t\to1} \end{aligned} \tag{3}$$

$V_{t\leftarrow0}$ is a weight matrix used to combine $g(I_0, F_{t\to0})$ and $g(I_1, F_{t\to1})$. As mentioned earlier, in contrast to the pixels in images, our 2D packet data pixels represent distances, which means their values will change in dynamic scenarios. Optical flow alone is not enough to interpolate the enclosed B frames; pixel values also need to be approximated from both $g(I_0, F_{t\to0})$ and $g(I_1, F_{t\to1})$. Assuming that a pixel's value changes linearly between two I frames, we restrict the elements of $V_{t\leftarrow0}$ to a range from 0 to 1 and we have:

$$V_{t\leftarrow0} = 1 - V_{t\leftarrow1}. \tag{4}$$

Finally, we can obtain our interpolated B frame $\hat{B}_t$:

$$\hat{B}_t = \frac{1}{Z} \odot \big((1-t)V_{t\leftarrow0} \odot g(I_0, F_{t\to0}) + tV_{t\leftarrow1} \odot g(I_1, F_{t\to1})\big),$$

where $Z = (1-t)V_{t\to0} + tV_{t\to1}$ is a normalization factor and $\odot$ denotes element-wise multiplication, implying content-aware weighting of the input.

B. NETWORK DETAIL

Fig 3 shows in detail the flow computation network and the flow interpolation network. Note that both networks are U-nets and have the same architecture. The only differences are the number of input and output channels.

U-net consists of an encoder with a decoder. Since pixels may move long distances in driving scenarios, we use a filter with a large kernel size in the first few layers of the encoder to capture this long range motion.

C. PRE-PROCESSING

Before sending the 2D packet data into the network, we first perform two pre-processing steps. The first step is a data normalization operation and the second step is a padding operation to alleviate the effect of signal loss in the LiDAR scanner.

1) DATA NORMALIZATION

Before sending data into the network, it's important to normalize the data in order to improve the performance of the network. There are many different methods for data normalization. For images, normalizing an RGB color image G into $G/255 - 0.5$ allows the data to range from 0 to 1 and to be zero-centered, which is a good choice under the hypothesis that all of the colors appear at about the same frequency, on average. LiDAR packet data in a 2D format is very different however, because of the long detection range and high accuracy of LiDAR scanners, which means each pixel's value will have a much larger range of possible values than a color image. For example, most LiDAR scanners manufactured by Velodyne use 2 bytes of data to store the distance information for each point, which means the distance values can range from 0 to 65,535. At the same time, the distribution of distance values is unique and highly dependent on the driving scenario. Generally speaking, today's 3D LiDAR scanners can detect objects over 100 meters away, and even up to 300 meters away, although in real world applications most of the reflection points obtained are 50 meters away or closer. To normalize the data well before training, we randomly choose packet data from 100 frames, calculate the mean μ and create a histogram of the distance values. By setting threshold θ so that 95% of the distance values fall under this value, we can normalize each packet data matrix P by calculating $(P - \mu)/\theta$.

2) PADDING

The padding operation described here is an original operation which we propose in this paper for the first time, and is used to facilitate the training of our deep learning network with LiDAR packet data in a 2D format. Signal loss is a big issue when using LiDAR sensors. LiDAR sensors scan the surrounding environment by emitting laser pulses and collecting the reflected signals. Sometimes, because of the angle of the pulse, the material reflecting the pulse or the color of an obstacle, etc., the reflected signal will be so weak that it cannot be analyzed properly and the correct distance cannot be calculated. In such cases, the reflection signal will result in a 0 value. Fig 4 shows an example of this phenomenon.

These signal losses cannot be foreseen, and are equivalent to the random deletion of some of the reflection values, which compromises the data's structure and significantly misleads the network during training. At the same time, it should be noted that not all 0 values are caused by information loss. If there is no obstacle in the path of the laser pulse, the reflection value will also be 0, thus these 0 values are regarded as normal values.

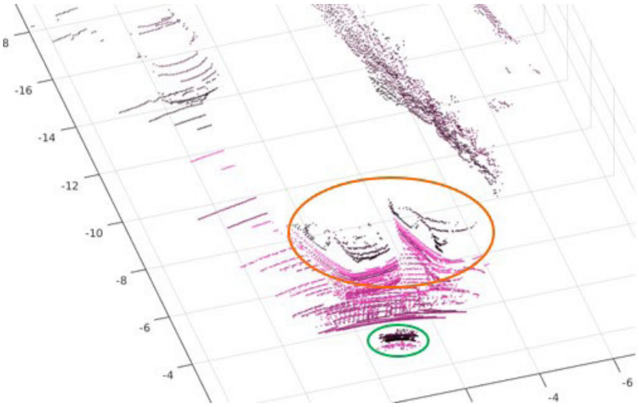


FIGURE 4. Example of signal loss phenomenon. Different colors in the point clouds represent points detected by different laser emission modules of the LIDAR scanner. Points in the green ellipse are 0 value reflection signals after calibration, which are supposed to be located in orange ellipse to add detail to the detected vehicles.

Algorithm 1 Preprocessing Step 2: Padding

```
Input: $m$ by $n$ packet data matrix $P$
Output: $m$ by $n$ packet data matrix $P'$
for $i = 1$ to $m$ do
    for $j = 1$ to $n$ do
        if $(P(i, j) = 0)$ then
            if $(j > 1)$ then
                $P'(i, j) = P(i, j - 1)$
            else
                $P'(i, j) = P(i - 1, j)$
            end if
        else
            $P'(i, j) = P(i, j)$
        end if
    end for
end for
return $P'$
```

To alleviate the effect of 0 values caused by signal loss while retaining the normal 0 values, we propose an original padding operation to estimate the lost signals:

As a result, the normal 0 values, which are usually densely concentrated and generate 0 value zones, are mostly retained since our padding technique only alters the first column of a 0 value zone and retains the rest of the 0 values. However, for 0 values caused by signal loss, which usually appear sporadically, our padding operation results in a reasonable, estimated value, namely $P(i, j - 1)$, which is the value of the previous reflected emission from the same laser emitter. Note that during training we pad both the I frames and B frames (as ground truth), while during compression we only pad the I frames when predicting the B frames.

This padding operation can significantly accelerate network convergence during training and lead to better compression performance, as will be described in Section VI-D.

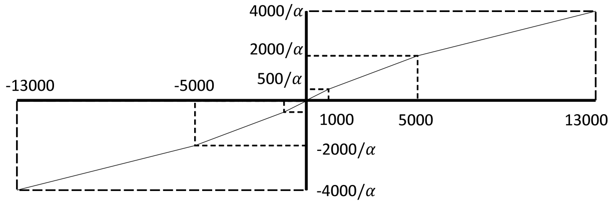


FIGURE 5. **Quantization of I frame and residual difference values. Up to three steps are used during quantization, depending on the size of the absolute difference value. After quantization, original data ranging from −13000 to 13000 cm are mapped to continuous integers ranging from −4000/α to 4000/α. Parameter α controls the range after quantization and can also be used to tune the compression ratio and loss.**

V. CODING MODULE

During coding, the 2D matrix of I frames I_0, I_1 derived from LIDAR packet data, and the residual R_t between $\hat{B}_t$ and B_t, are quantized and coded. Matrix I_0, I_1 is then losslessly compressed using a JPEG based compression method as described in detail in a previous study [3].

To encode residual R_t, we quantize each pixel in R_t as shown in Fig. 5 and then JPEG-LS [30] is used to encode the quantized values, allowing spatial redundancy to be further reduced. Thus, after interpolating B frames from I frames to reduce temporal redundancy, and utilizing the 2D characteristics of the raw scanner data to reduce spatial redundancy, compressed streaming point cloud data is obtained.

VI. EVALUATION

A. TRAINING & TEST DATA

To train our network, we used 55,961 frames of point cloud data as training data, consisting of 10 Hz driving data from various areas of Japan, which included different situations such as urban roads, highways, etc. All of the point cloud data was collected by Velodyne HDL-64ES2 or S3 sensors, and is from the TierIV open dataset https://rosbag.tier4.jp/.

In order to evaluate the performance of the proposed compression method and compare it with other methods, the use of an open dataset is highly desirable. Some point cloud compression methods employ the fr1/room dataset [31], while in the autonomous driving domain the KITTI dataset [32] is often used. Unfortunately, the KITTI dataset does not provide the raw LiDAR packet data which our proposed method requires to function. Therefore, we used our own test data, obtained using a Velodyne HDL-64E S2 sensor, for our comparative evaluation, which can be downloaded here: https://drive.google.com/drive/u/0/folders/1qUG_kEqfoT3oCOIMHI0FOktY8Nk9hC3b. The data contains three, one-minute sets of driving data representing three different driving scenarios; exiting a parking lot, driving in a residential area and driving on a major urban road.

B. IMPLEMENTATION

In our interpolation module, we set parameter n to $n = 3$, which means we are interpolating three B frames between each pair of I frames. During training, our loss function l_r represents the L1 loss between the interpolated B frames and true B frames:

$$l_r = \frac{1}{n} \sum_{i=1}^{n} \|\hat{B}_{t_i} - B_{t_i}\|_1. \tag{5}$$

In the coding module, we use parameter $\alpha = (\frac{3}{4})^i \quad i \in [0, 1, 2, \ldots, 11]$ to tune the trade-off between compression rate vs. information loss

C. VISUALIZATION OF INTERPOLATION RESULTS

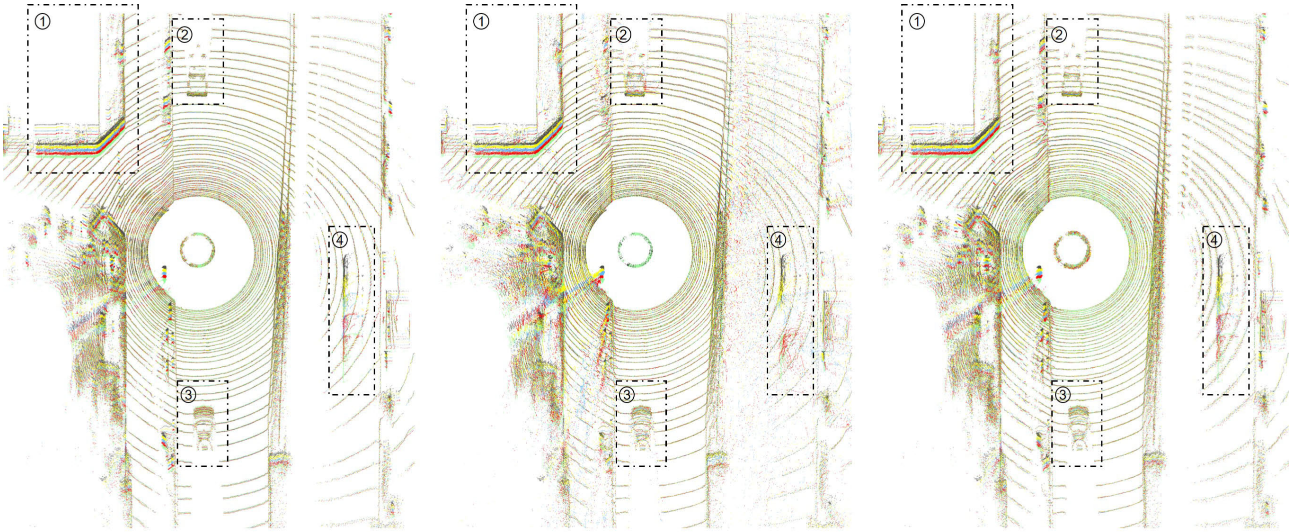
In contrast to video interpolation tasks, which usually interpolate frames over a very short period, such as 1/30th or 1/60th of a second, our interpolation covers a much longer time period ($1/n$ second), or 1/3 of a second in this case. As mentioned in the introduction, we do not expect our proposed method to perfectly predict LiDAR B frames.

In Fig 6 (b), we stack the point clouds of a set of predicted B frames with the point clouds of the corresponding I frames and visualize them. Since all of the frames use the LiDAR apparatus as the origin point of their coordinates, when the vehicle moves, in our visualizations the static environment moves inversely, but the center of our figure (i.e., the location of the LiDAR unit) remains static. By comparing the point clouds of the predicted B frames 6 (b) with point clouds of the true B frames 6 (a), we can see that our interpolation network efficiently predicted most of the reflected points from the ground and key parts of the static environment, such as the points in dotted line box (1). This indicates that our interpolation network was able to accurately predict motion in the B frames between each pair of I frames. Our interpolation network can also predict the motion of dynamic objects, such as the vehicle in dotted line box (2), which is relatively static in relation to our LiDAR unit, and the motion of the vehicle in dotted line box (3), which is moving in relation to the ego vehicle. In some regions we can see that our interpolation network failed and lost some detail, for example the vehicle in dotted line box (4).

By storing these information losses, i.e., the residuals between the predicted and true B frames, in the encoder module, we can obtain accurate decompression output as 6 (c) shows.

To examine the performance of our interpolation method in more detail, Fig 7 visualizes the point cloud of a single predicted B frame and compares it with its corresponding true B frame.

As we mentioned earlier, our U-net based interpolation method can predict quite well most of the important details from the pairs of surrounding point cloud reference frames, such as reflections from the ground, other vehicles and walls. Note also that even some irregular textures can be predicted, such as the points in the red boxes. However, our interpolation method was unable to accurately predict objects in vacant spaces, such as those shown in purple ellipses. These blank areas correspond to large gaps between the values of adjacent



(a) a pair of I frames with true B frames

(b) a pair of I frames with B frames predicted by interpolation

(c) a pair of I frames with predicted B frames + quantized residual (i.e. decompression) at 5.3 cm SNNRMSE

FIGURE 6. Visualizing a set of stacked "IBBBI" frames. Points in the five frames are colorized using black (I), yellow (B), blue (B), red (B) and green (I), respectively, representing motion over time from the black I frame to the green I frame. For a detailed definition of SNNRMSE (Symmetric Nearest Neighbor Root Square Mean Error) please see Section VI-D.

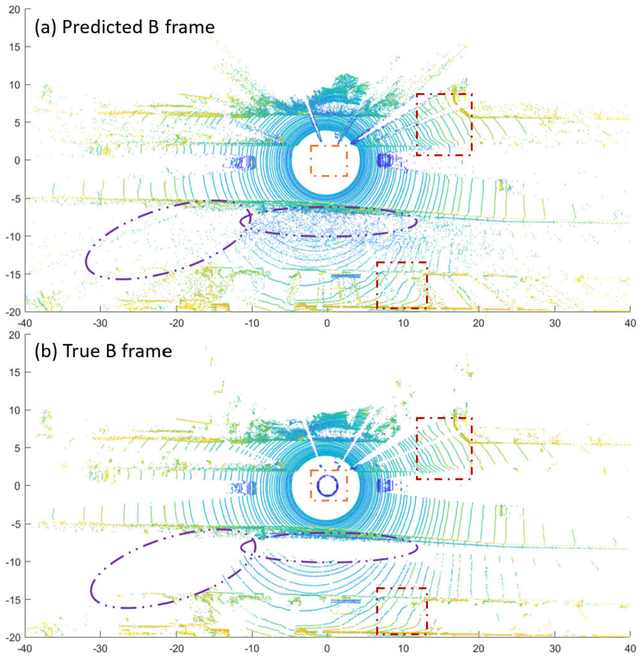


FIGURE 7. Comparison of the point clouds of a predicted and true B frame, colorized by intensity.

pixels in the 2D packet data matrix. Sometimes, in these types of areas, our interpolation network does not predict the motion (optical flow) of pixels well, which leads to incorrect pixel (i.e., distance) values. This is also the reason why we could not accurately predict the motion of the vehicle in dotted line box (4) in Fig 6.

One more thing which needs to be noted is that the points in the orange box at the center of Fig 7 (b), which form a circle, are 0 value reflections after calibration. Because of the padding operation we performed during pre-processing, in predicted B frame (a) we do not have this "circle", and instead these points from 0 value reflections are possibly distributed at their correct locations.

D. COMPARISON OF PROPOSED METHOD WITH OTHER COMPRESSION METHODS

In this section, we compare our proposed method with other streaming point cloud compression methods. Our previously proposed MPEG [3] and SLAM [25] based compression methods also utilized LiDAR packet data to reduce spatial redundancy, but in those studies we used a completely different strategy to reduce temporal redundancy. The octree compression method [23] uses a double octree to deal with spatial and temporal redundancy in point cloud data, and is probably the most popular generic point cloud compression method.

In our comparison, we mainly considered two technical indexes used to evaluate the performance of compression methods: 1) processing time, and 2) bitrate (volume after compression) vs. information loss.

Processing time is an important metric when evaluating a compression method. In our method, interpolation module, which is essentially a stack of convolutional layer, costs most computation. Its time complexity is:

$$O\left(\sum_{l=1}^{d} n_{l-1} \cdot s_l^2 \cdot n_l \cdot m_l^2\right) \tag{6}$$

Here l is the index of a convolutional layer, and d is the depth (number of convolutional layers). n_l is the number of filters (also known as "width") in the l-th layer. n_{l-1} is also known as the number of input channels of the l-th layer. s_l is the spatial size (length) of the filter. m_l is the spatial size of the output feature map [33].

Take point cloud from velodyne HDL-64ES2 sensor, which contains 133,632 points one scan, for example. Using an Intel Core i7-7820X CPU with a GeForce GTX 1080 GPU (used only for U-net processing), on average the MPEG based compression method took 2.215 seconds of processing time to compress one frame, while the SLAM based method needed 3.624 seconds per frame. Considering that point cloud data from LiDAR scanners is usually collected at 10 Hz, only our U-net based compression method (0.089 seconds per frame) and octree compression (0.0241 seconds per frame) can be implemented in real time, which is a major advantage.

To quantitatively evaluate the compression performance of various approaches, we use bits per point (bpp) to measure data volume after compression and Symmetric Nearest Neighbor Root Mean Squared Error (SNNRMSE) to measure information loss after decompression.

SNNRMSE is the conventional criterion used to quantify the difference between two point cloud frames [11]. Given two point cloud frames P and Q, for each point p in P find the closest point q in Q (in Euclidean distance), where q is defined as $q = \mathrm{NN}(p, Q)$.

$$\mathrm{MSE_{NN}}(P, Q) = \sum_{p \epsilon P} (p - q)^2 / |P| \tag{7}$$

Here $|P|$ represents the number of points in P. Then:

$$\mathrm{RMSE_{NN}}(P, Q) = \sqrt{\mathrm{MSE_{NN}}(P, Q)} \tag{8}$$

By considering both $\mathrm{RMSE_{NN}}(P, Q)$ and $\mathrm{RMSE_{NN}}(Q, P)$, $\mathrm{RMSE_{SNN}}(P, Q)$ can be calculated as follows:

$$\mathrm{RMSE_{SNN}}(P, Q) = \sqrt{0.5\mathrm{MSE_{NN}}(P, Q) + 0.5\mathrm{MSE_{NN}}(Q, P)} \tag{9}$$

The original point cloud required 192 bpp. For both bpp and SNNRMSE, the smaller the value the better.

Fig 8 shows bpp vs. SNNRMSE for various compression methods. When comparing our proposed method without padding (grey dotted line) with octree, SLAM based and MPEG based compression methods, our method achieved the best performance within a certain range, for example from 2 bpp to 4.5 bpp in the parking lot scenario.

However, thanks to our innovative padding operation during pre-processing, the performance of our proposed method is greatly improved. Our proposed approach with padding (black line) performs much better that the octree and SLAM based compression methods, except when very low SNNRMSE is required (around 2 cm). Fortunately, as a lossy compression, we usually can bear much more than this level. In fact, 2 cm is the sensor accuracy limit of the LiDAR scanner used in the experiment. The MPEG based method sacrifices accuracy for high compression. Even so, our proposed method (black line) achieves a lower bitrate at around 10 to 12 cm SNNRMSE. In addition, the proposed method can provide more accurate data after decompression, which is more useful in our experience, because when SNNRMSE is over 10 cm, the decompressed data is already obviously different visually from the original.

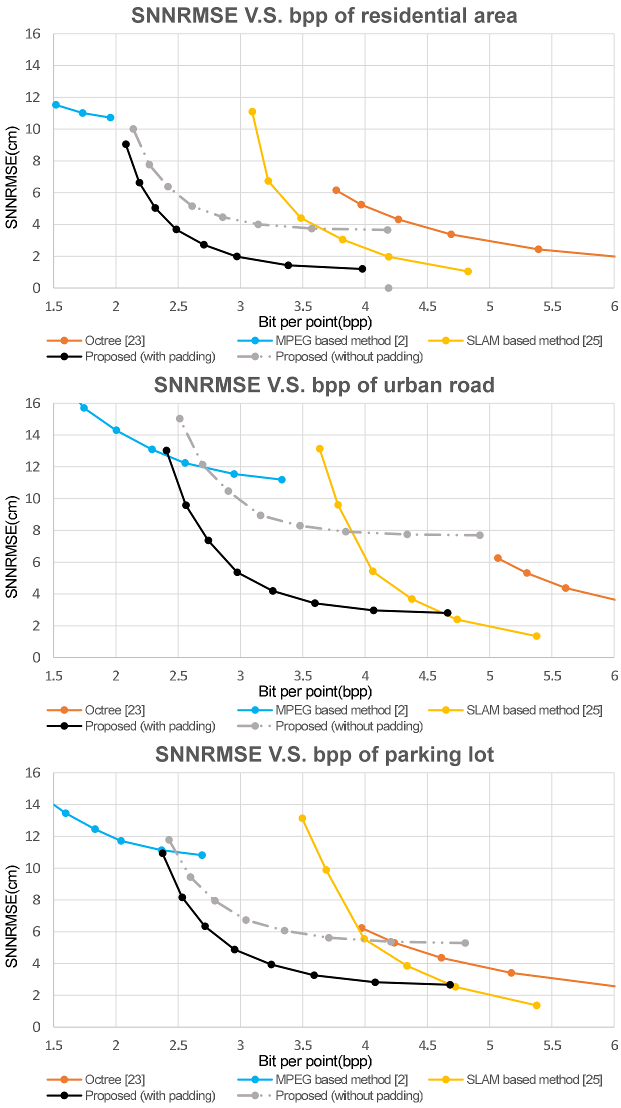


FIGURE 8. **SNNRMSE vs. Bitrate (in bits per point) for our proposed U-net based streaming point cloud compression method (with and without padding), in comparison with other compression methods.**

VII. CONCLUSION

In this paper, we proposed a deep learning-based, streaming point cloud compression method which can be performed in real time. By using u-nets and pre-processing, we demonstrated that we can efficiently reduce temporal redundancy. Spatial redundancy was also reduced by reformatting raw LiDAR packet data into a 2D matrix and using JPEG-LS for data encoding. Our experimental comparison with other streaming point cloud compression methods, such as octree [23], MPEG [3] and SLAM [25] based methods, showed that our proposed compression method can achieve a greater compression rate with less information loss.

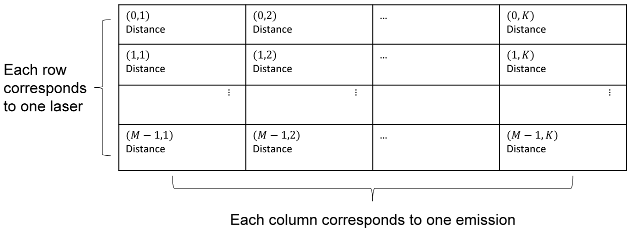


FIGURE 9. **Arranging raw packet data into a 2D matrix. Each row corresponds to one laser ID, and each column corresponds to one emission. The value of each pixel represents the distance information.**

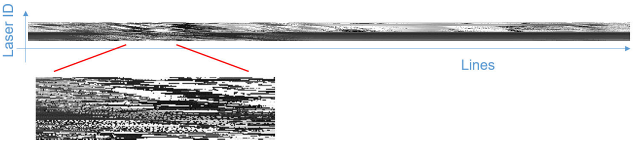


FIGURE 10. **Visualization of raw packet data in an image-like format. A pixel's grayscale value from black to white represents a distance from 0 to 13,000 cm. Noted that without calibration, we could not understand the raw data the way we can understand a depth map.**

APPENDIX

POTENTIAL 2D CHARACTERISTICS OF POINT CLOUD DATA COLLECTED BY LiDAR SCANNERS

Each point in a set of point cloud data is represented by x, y, and z coordinates. However, raw packet data represents each point using a distance value, a rotation angle and a laser ID. Rotation angle here means yaw angle of LiDAR and all of a particular laser's emissions occur at the same yaw angle once without taking calibration into consideration. Laser ID here represents pitch angle information. In LiDAR systems, every laser sensor is fixed at a specific location and angle, so that if the laser ID is known we can easily determine the pitch angle of the beam. In other words, raw packet data can be roughly thought as a kind of polar-coordinate-like representation of 3D point cloud. After a calibration process $f(R) = P$, which uses a calibration file to correct angle, distance and starting locations, raw packet data R can be converted into a point cloud P in real-time.

By arranging raw packet data into a 2D matrix as shown in Fig. 9, the information can be stored losslessly within a 2D matrix. Fig. 10 shows an example of an image created using this type of raw packet data. Laser ID information is represented by row number i; while information about the rotation angle, which corresponds to column j, could be coded into a few bits of data because, generally, the difference between adjacent rotation angles is uniform. As a result, by using run length coding all of the rotation angle information can be efficiently captured. These potential 2D characteristics of point clouds are in fact based on LiDAR's own operating principle.

We need to store three elements, x, y and z coordinates, for each point in a point cloud, but when utilizing the data in this image-like format, for the most part we only need one element, distance from the scanner to the detected point. Thus, the 2D format is itself already a kind of data compression.

REFERENCES

[1] H. Jiang, D. Sun, V. Jampani, M.-H. Yang, E. Learned-Miller, and J. Kautz, "Super SloMo: High quality estimation of multiple intermediate frames for video interpolation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 9000–9008.

[2] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, M. Sokolsky, G. Stanek, D. Stavens, A. Teichman, M. Werling, and S. Thrun, "Towards fully autonomous driving: Systems and algorithms," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2011, pp. 163–168.

[3] C. Tu, E. Takeuchi, C. Miyajima, and K. Takeda, "Compressing continuous point cloud data using image compression methods," in *Proc. IEEE Int. Conf. Intell. Transp. Syst. (ITSC)*, Nov. 2016, pp. 1712–1719.

[4] J. Peng, C.-S. Kim, and C.-C. J. Kuo, "Technologies for 3D mesh compression: A survey," *J. Vis. Commun. Image Represent.*, vol. 16, no. 6, pp. 688–733, Dec. 2005.

[5] M. Pauly and M. Gross, "Spectral processing of point-sampled geometry," in *Proc. ACM 28th Annu. Conf. Comput. Graph. Interact. Techn.*, 2001, pp. 379–386.

[6] E. Hubo, T. Mertens, T. Haber, and P. Bekaert, "Self-similarity based compression of point set surfaces with application to ray tracing," *Comput. Graph.*, vol. 32, no. 2, pp. 221–234, 2008.

[7] T. Ochotta and D. Saupe, "Compression of point-based 3D models by shape-adaptive wavelet coding of multi-height fields," in *Proc. 1st Eurographics Conf. Point-Based Graph.*, 2004, pp. 103–112.

[8] T. Ochotta and D. Saupe, "Image-based surface compression," *Comput. Graph. Forum*, vol. 27, no. 6, pp. 1647–1663, 2008.

[9] R. Schnabel, S. Möser, and R. Klein, "A parallelly decodeable compression scheme for efficient point-cloud rendering," in *Proc. Symp. Point Based Graph. (SPBG)*, 2007, pp. 119–128.

[10] R. Schnabel, S. Möser, and R. Klein, "Fast vector quantization for efficient rendering of compressed point-clouds," *Comput. Graph.*, vol. 32, no. 2, pp. 246–259, 2008.

[11] T. Golla and R. Klein, "Real-time point cloud compression," in *Proc. IEEE Intell. Robots Syst. (IROS)*, Sep./Oct. 2015, pp. 5087–5092.

[12] V. Morell, S. Orts, M. Cazorla, and J. Garcia-Rodriguez, "Geometric 3D point cloud compression," *Pattern Recognit. Lett.*, vol. 50, pp. 55–62, Dec. 2014.

[13] H. Houshiar and A. Nüchter, "3D point cloud compression using conventional image compression for efficient data transmission," in *Proc. 25th Int. Conf. Inf., Commun. Automat. Technol. (ICAT)*, Oct. 2015, pp. 1–8.

[14] K. Kohira and H. Masuda, "Point-cloud compression for vehicle-based mobile mapping systems using portable network graphics," *ISPRS Ann. Photogramm., Remote Sens. Spatial Inf. Sci.*, vol. 4, no. 2, pp. 99–106, 2017.

[15] H. Yin and C. Berger, "Mastering data complexity for autonomous driving with adaptive point clouds for urban environments," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2017, pp. 1364–1371.

[16] G. Toderici, D. Vincent, N. Johnston, S. J. Hwang, D. Minnen, J. Shor, and M. Covell, "Full resolution image compression with recurrent neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 5306–5314.

[17] C. Tu, E. Takeuchi, A. Carballo, and K. Takeda, "Point cloud compression for 3D LiDAR sensor using recurrent neural network with residual blocks," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2019, pp. 3274–3280.

[18] J. Hayes, "LIDAR point cloud compression," U.S. Patent 9 753 124 B2, Sep. 5, 2017.

[19] S. Gumhold, Z. Kami, M. Isenburg, and H.-P. Seidel, "Predictive point-cloud compression," in *Proc. ACM SIGGRAPH Sketches*, 2005, p. 137.

[20] E. Hubo, T. Mertens, T. Haber, and P. Bekaert, "The quantized kd-tree: Efficient ray tracing of compressed point clouds," in *Proc. IEEE Symp. Interact. Ray Tracing*, Sep. 2006, pp. 105–113.

[21] R. Schnabel and R. Klein, "Octree-based point-cloud compression," in *Proc. Eurographics Symp. Point-Based Graph. (SPBG)*, vol. 6, Jul. 2006, pp. 111–120.

[22] J. Elseberg, D. Borrmann, and A. Nüchter, "One billion points in the cloud—An octree for efficient processing of 3D laser scans," *ISPRS J. Photogramm. Remote Sens.*, vol. 76, pp. 76–88, Feb. 2013.

[23] J. Kammerl, N. Blodow, R. B. Rusu, S. Gedikli, M. Beetz, and E. Steinbach, "Real-time compression of point cloud streams," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2012, pp. 778–785.

[24] D. Thanou, P. A. Chou, and P. Frossard, "Graph-based compression of dynamic 3D point cloud sequences," *IEEE Trans. Image Process.*, vol. 25, no. 4, pp. 1765–1778, Apr. 2016.

[25] C. Tu, E. Takeuchi, C. Miyajima, and K. Takeda, "Continuous point cloud data compression using SLAM based prediction," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2017, pp. 1744–1751.

[26] S. Niklaus, L. Mai, and F. Liu, "Video frame interpolation via adaptive separable convolution," in *Proc. IEEE Int. Conf. Comput. Vis. (CVPR)*, Oct. 2017, pp. 261–270.

[27] S. Niklaus and F. Liu, "Context-aware synthesis for video frame interpolation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 1701–1710.

[28] T. Zhou, S. Tulsiani, W. Sun, J. Malik, and A. A. Efros, "View synthesis by appearance flow," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*. New York, NY, USA: Springer, 2016, pp. 286–301.

[29] Z. Liu, R. A. Yeh, X. Tang, Y. Liu, and A. Agarwala, "Video frame synthesis using deep voxel flow," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 4463–4471.

[30] M. J. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I lossless image compression algorithm: Principles and standardization into JPEG-LS," *IEEE Trans. Image Process.*, vol. 9, no. 8, pp. 1309–1324, Aug. 2000.

[31] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems," in *Proc. IEEE Intell. Robots Syst. (IROS)*, Oct. 2012, pp. 573–580.

[32] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *Int. J. Robot. Res.*, vol. 32, no. 11, pp. 1231–1237, 2013.

[33] K. He and J. Sun, "Convolutional neural networks at constrained time cost," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 5353–5360.

CHENXI TU received the B.S. degree from the School of Information and Communication Engineering, Beijing University of Posts and Telecommunications, Beijing, China, in 2013, and the M.S. degree in information science from Nagoya University, Nagoya, Japan, in 2016, where he is currently pursuing the Ph.D. degree in informatics. His research focuses on point cloud data processing.

EIJIRO TAKEUCHI received the bachelor's, master's, and Ph.D. degrees from the Intelligent Robot Laboratory, University of Tsukuba, Japan. From 2008 to 2014, he was with Tohoku University, Japan, as an Assistant Professor. Since 2014, he has been with Nagoya University, Japan, where he is currently an Associate Professor with the Graduate School of Informatics. His main researches focus on localization, mapping in robotics, and autonomous driving.

ALEXANDER CARBALLO received the Dr. Eng. degree from the Intelligent Robot Laboratory, University of Tsukuba, Japan. From 1996 to 2006, he was a Lecturer with the School of Computer Engineering, Costa Rica Institute of Technology. From 2011 to 2017, he was involved in research and development at Hokuyo Automatic Company Ltd. Since 2017, he has been a Designated Assistant Professor with the Institutes of Innovation for Future Society, Nagoya University, Japan. His main research interests include LiDAR sensors, robotic perception, and autonomous driving.

KAZUYA TAKEDA received the B.E.E., M.E.E., and Ph.D. degrees from Nagoya University, Japan. Since 1985, he has been with Advanced Telecommunication Research Laboratories and with KDD R&D Laboratories, Japan. In 1995, he started a Research Group for Signal Processing Applications at Nagoya University, where he is currently a Professor with the Institutes of Innovation for Future Society. His main researches focus on investigating driving behavior using data centric approaches and utilizing signal corpora of real driving behavior. He is a Senior Member of the IEEE. He is also serving as a member of the Board of Governors of the IEEE ITS Society.