RIDDLE: Lidar Data Compression with Range Image Deep Delta Encoding

Xuanyu Zhou* Charles R. Qi* Yin Zhou Dragomir Anguelov
Waymo LLC

Abstract

Lidars are depth measuring sensors widely used in autonomous driving and augmented reality. However, the large volume of data produced by lidars can lead to high costs in data storage and transmission. While lidar data can be represented as two interchangeable representations: 3D point clouds and range images, most previous work focus on compressing the generic 3D point clouds. In this work, we show that directly compressing the range images can leverage the lidar scanning pattern, compared to compressing the unprojected point clouds. We propose a novel data-driven range image compression algorithm, named RIDDLE (Range Image Deep DeLta Encoding). At its core is a deep model that predicts the next pixel value in a raster scanning order, based on contextual laser shots from both the current and past scans (represented as a 4D point cloud of spherical coordinates and time). The deltas between predictions and original values can then be compressed by entropy encoding. Evaluated on the Waymo Open Dataset and KITTI, our method demonstrates significant improvement in the compression rate (under the same distortion) compared to widely used point cloud and range image compression algorithms as well as recent deep methods.

1. Introduction

Lidar (or LiDAR, short for light detection and ranging) sensors are commonly used in applications that require 3D scene understanding such as autonomous driving and augmented reality. However, with the growing resolution of lidars, storing and transmitting large volumes of sequential lidar data become a challenge. There is a strong need to develop effective algorithms for lidar data compression.

While the measurements of a lidar scan are often used as a 3D point cloud, the raw lidar data can be represented as a more structured format: a range image, where each pixel corresponds to a laser shot, each row represents shots from the same laser, each column represents shots at a specific azimuth rotation angle. Given the lidar scanning mechanism (directions of the lasers) and sensor poses (6D poses in the global coordinate at the timestamp of every shot), a range image and its corresponding point cloud can be converted interchangeably and losslessly. By organizing the points in a range image, instead of storing the *three*-dimensional coordinates of the points, we can just store *one*-dimensional ranges (around 3x saving in storage). Given this observation, in contrast to previous works that focus on compressing 3D point clouds [9, 16, 23], we propose to directly compress range images to leverage the lidar scanning patterns.

As range images are in the image format, naturally we can apply existing compression methods for optical images (RGB or grayscale); however, those methods have their limitations. For example, the PNG format is often used to compress depth images in indoor datasets [4, 11, 25], where the depth value are normalized and quantized to 16-bit integers and compressed losslessly. While PNG also applies to compress lidar range images, it is not data-driven and does not use temporal information. There are also attempts to use auto-encoder networks [31] to lossily compress range images by storing the bottleneck layer output. However, as range values often have a much wider distribution than RGB colors, it is challenging to learn an accurate reconstruction, especially at the object boundaries.

In this work, we propose *RIDDLE* (Range Image Deep DeLta Encoding), a data-driven algorithm to compress range images with predictive neural networks (Fig. 2). Our method is inspired by the use of delta encoding in PNG image compression. However, instead of simply computing a difference between close-by pixels, we adopt a deep model to predict the pixel value from context pixels. The deep model takes a local patch of the decoded range image and predicts the attributes of the next pixel in a raster-scanning order (a similar process to the sequential image decoder PixelCNN [33]). We can then entropy encode the residuals between the predicted values and the original values to achieve *lossless* compression under a chosen quantization rate. In this scheme, the more accurate the prediction is, the smaller the entropy of the residuals are – improving the compression rate is equivalent to developing a more accurate predictive model.

What is unique in our model design is that we represent

*equal contribution

local image patches as point clouds in the *spherical* coordinates (with azimuth, elevation and range values) to reflect the non-uniform ray angles of each shot (or pixel), which lifts the 2D pixels to 3D point clouds. By further lifting the 3D points to 4D with a timestamp channel, we can unify the way we represent context pixels/points from both the *current* and *history* scans. Since our model directly takes in point clouds, neither interpolation (to the image grid) nor image cropping (projected points from history frames may span different image regions) is needed. On the other hand, as to the model output formulation, instead of directly regressing the pixel values (which is often multi-modal), we treat each pixel in the input patch as an anchor and predict a confidence score as well as a residual value per anchor.

Evaluated on the large-scale Waymo Open Dataset (WOD) [26], we show that our method reduces the bitrate by more than 65% for the same distortion (measured using the point-to-point Chamfer distance) or reducing more than 85% distortion for the same bitrate, compared to the MPEG standard compression method G-PCC [14] while also significantly outperforming other baselines like Draco [1] and PNG. On the KITTI dataset [13], we compare with prior art deep compression methods (using octrees) and show our method has a clear advantage over them, thanks to its use of the range image representation and the accurate prediction model. We also evaluate the impact of compression on downstream perception tasks such as 3D object detection and provide extensive ablation studies to validate our design choices.

2. Related Work

Point cloud compression As 3D applications rise, recent years have seen an increasing number of algorithms proposed for point cloud compression. One family of the methods uses octrees to represent and compress quantized point clouds [10, 12, 24]. The Motion Picture Experts Group (MPEG) has released a related point cloud compression (PCC) standard, called geometry-based PCC (G-PCC) [14], using the octree structure and various ways to predict the next-level content. More recently, Octsqueeze [16] was proposed to use a neural network as a conditional entropy model to estimate the octree occupancy symbols, and MuSCLE [9] extends it by including temporal prior from previous frames. VoxelContextNet [23] further leverages the voxel context for the octree structure prediction. These neural network-based methods consistently show improvements over G-PCC which uses hand-crafted entropy models. While the octree-based methods are flexible to model arbitrary point clouds (from either a lidar sensor or multi-view reconstruction), they do not make use of the point distribution patterns in lidar range images.

As a lidar point cloud can be represented as a range image, image-based compression methods can be adapted for its compression. For example, [3, 7, 15] applied traditional image compression methods such as JPEG, PNG and TIFF to compress the range images. A sequence of range images could be seen as a video, and video-based compression method like H.264 was applied to compress lidar sequences [20]. MPEG also proposed a PCC (V-PCC) standard that compresses dynamic point clouds via HEVC video codex [14]. Our work extends them to leverage deep models and delta encoding to compress range images.

Auto-encoders have been used to achieve lossy compression of point clouds. [34, 35] proposed to train an encoder-decoder point cloud reconstruction network and entropy encode the bottleneck layer as the compressed data. Similarly, [31] trained an auto-encoder to reconstruct range images and compress the bottleneck vectors. While these methods may achieve high compression rates, the reconstructed point clouds could have strong artifacts, especially at the object boundaries resulting in unbounded errors in the lossy compression scheme.

Learned image and video compression Image and video compression are well-studied fields with many standards (for example: PNG, JPEG, TIFF for images, H.264 and HEVC for videos). Among them, PNG is highly related to our work as it uses lossless image compression using delta encoding. With the popularity of deep convolutional neural networks for image understanding, deep model-based image and video compression have also been widely explored [5, 6, 18, 19, 29, 30]. Many of them leverage an encoder-decoder neural network (for example, a variational auto-encoder [5]) for the compressing (encoding the image to a latent vector) and decompressing (decode/generate the image from the vector). For the decoding architectures, sequential models such as PixelCNN [21] and PixelRNN [33] inspired our predictive model design.

3. Problem Formulation

For most lidar sensors, one scan can be interchangeably represented as either a point cloud $P \in \mathbb{R}^{N \times C}$ or a range image $I \in \mathbb{R}^{H \times W \times C}$, where N is the number of points, H and W are height and width of the range image (H is the number of laser beams in the lidar and W is the number of shots per laser per frame), C is the feature dimension for each point. Each valid pixel in the range image represents a laser shot corresponding to one point in the point cloud. The channels include the range value and other attributes such as reflection intensity. The conversion rule between a point cloud and a range image depends on the laser scanning mechanism (the laser shot azimuth and elevation angles) as well as the sensor poses (the 6D pose of the laser sensor at the time of each laser shot), as illustrated in Fig. 1.

Specifically, in a range image I, given a pixel location (i, j) (which maps to a specific laser shot angle) and its

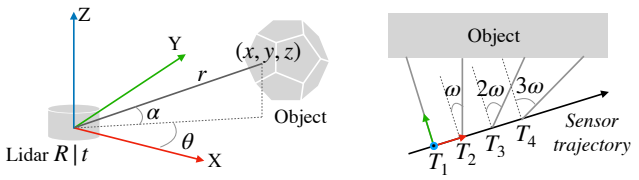

Figure 1. **Illustration of laser shots.** *Left:* A single laser shot. *Right:* Laser shots across time (in a bird's eye view). We show four consecutive laser shots (with delta azimuth angle ω) that measure the ranges from the (moving) sensor to the object. To convert the range values to a point cloud, we need to know the ranges, the shot angles, as well as the sensor poses at each shot.

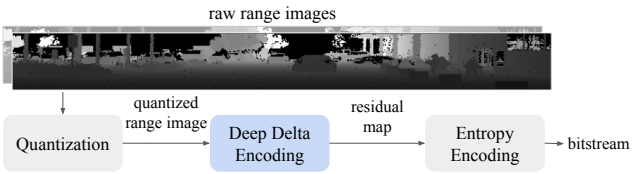

Figure 2. **The deep delta encoding pipeline for lidar range image compression.** Given a lidar range image, we first quantize the attribute values and then run inference of the predictive model on the quantized range image to derive residuals. Finally we use entropy encoders to compress the residuals to a bitstream.

range value, we get a laser measurement (r, θ, α) where r is the range value, θ (azimuth or yaw) and α (elevation or pitch) are the shot angles relative to the lidar sensor coordinate. The measurement can be converted to a point p in the sensor coordinate by:

$$p = (x, y, z) = (r\cos\alpha\cos\theta, r\cos\alpha\sin\theta, r\sin\alpha) \quad (1)$$

As at the time of each laser shot, the sensor pose $[R|t]$ (rotation and translation in the global coordinate) can be different (Fig. 1). To aggregate the shots into a point cloud, we need to convert the points to a shared global coordinate system to get the point set $P = \{R_i p_i^T + t_i\}, i = 1, ..., N$ where i is the index of the laser shot in a scan/range image.

Reversely, given the point cloud P of a scan (in the global coordinate), to convert it to the range image, we first need to transform *each point* to the sensor coordinate corresponding to its time of the shot. Then, we can easily get the (r, θ, α) by the reverse process of Eq. 1, which then maps back to the row and column indices.

For our lidar range image compression, we first quantize the range image I by rounding its pixel values to a predetermined quantization precision. Then our goal is to compress the quantized range image I' to a bitstream $b \in [0, 1]^n$ (with an n as small as possible), which can later be decompressed into the exact quantized range image I'. It is lossy with respective to the raw range image but *lossless* regarding the quantized range image.

Note that for calibrated lidars such as the ones used in the Waymo Open Dataset [26], each pixel in the range image corresponds to a fixed shot angle (θ, α) for the same lidar, so the angles do not need to be stored for the compression [1]. Besides, as sensor poses are often stored separately from range images and are shared with other modules (such as localization), we do not need to store sensor poses either. Only the range image needs to be compressed.

[1]For the main lidars used in WOD, pixel elevations are determined by the laser beam inclinations (64 numbers) and azimuths can be calculated based on uniform azimuth rotation. For other lidars such as Velodyne HDL-64, azimuth rotation angles are not uniform and need to be stored (one number for each column, costing only $\sim$ 0.1Kb per frame) [32].

4. Range Image Deep Delta Encoding

We first describe our overall compression pipeline in Sec. 4.1, then dive deep into the design of our prediction model in Sec. 4.2, and finally describe how we entropy encode the residuals in Sec. 4.3.

4.1. Pipeline Overview

As shown in Fig. 2, the input to our compression pipeline is a raw range image. First, we quantize the range image with a certain quantization precision (this allows us to store the deltas as discrete symbols). Next, the core part of the pipeline is the deep delta encoding. We train a deep model to predict the next pixel value in a raster scanning order. We then save the *delta* between the prediction (quantized) and the original (quantized) pixel value instead of saving the original pixel value. As the deltas are smaller and more concentrated in distribution than the original pixel values, they can be compressed more effectively. At the last step, the deltas (or the residual map) are entropy encoded to a compressed bitstream.

4.2. Deep Delta Encoding

Commonly used delta encoding adopts a linear prediction model to estimate the pixel values. In its simplest form, to predict a pixel $I_{i,j}$ at the i-th row and j-th column, its left pixel $I_{i,j-1}$ is used as the prediction. Other linear filters of left, up and nearby pixels can also be used. The *delta* between the prediction and the original pixel value is stored to be compressed. In our work, we propose to train a deep neural network to predict the pixel values and show that it can achieve significant improvement in prediction accuracy and compression rate. Next, we first introduce our model in its intra-prediction format (only using the information from the current frame/scan for the prediction) and then describe how we extend it to take temporal input from history scans. Please see the supplementary for more details on the model architecture, the losses and the training process.

Intra-frame Prediction Model Formally, the network models the conditional probability of the k-th pixel value (in the raster scanning order) conditioned on the quantized pixel values before k: $p(I_k;\Theta) = p(I_k|\{I'_{k-1},...,I'_1\};\Theta)$, where Θ are the network weights, I' is the quantized range image and I is the unquantized raw range image. Empirically, as shown in Fig. 3, instead of using the entire past context (e.g. with a RNN model), we can use local image patch of shape $h \times w$ as the context to predict the bottom right pixel of the patch, similar to the idea of the sequential image decoder PixelCNN [21].

Although the input to our network is an image patch, it is quite different from a typical RGB one. The relations of the range image pixels depend on the location of the patch and even the calibration of a specific lidar because the laser shot angles are often non-uniformly distributed. This is even more prominent in the inter-frame prediction when we re-project the points from history scans to the coordinate of the current shot. Therefore, we augment the range image with two extra channels: the delta azimuth and delta elevation angles relative to the angles of the to-be-predicted pixel, which lifts the 2D pixels to the 3D spherical coordinate. Furthermore, as range prediction is a geometry estimation problem, we found that empirically, using a 3D deep learning model such as PointNet [22] leads to more accurate prediction compared to using a 2D convolutional network.

As shown in Fig. 3, given the lidar calibration data, we first convert the range image patch to a mini point cloud (with maximally $hw-1$ points). Instead of directly regressing the pixel range value, which suffers from the uncertainty caused by the multi-modal distribution of attributes (esp. on the object boundaries), we formulate the prediction as an anchor-based classification and anchor-residual regression problem, where valid pixels in the range image patch are the anchors. The deep network predicts which pixel is the closest in value to the bottom right pixel and regresses a residual (it is an overloaded word here; it is different from the residual map in delta encoding) with respect to each anchor pixel.

Temporal Model The temporal model extends the intra-frame prediction model by leveraging contexts from both the current scan and the past scan. The point cloud representation (compared to the 2D pixel representation) enables us to unify the input from the past and current scans as we can represent all laser shots in the 4D (spherical plus time) coordinates.

Given the current scan (quantized) range image I'_T and the past scan range image I'_{T-1}, assume we want to predict the range value of pixel (i,j) in the current scan (k-th pixel in the raster scanning order). A naive baseline approach to use temporal data is to take the same neighborhood at that in I'_T (in terms of pixel rows and columns) from the last scan I'_{T-1} and concatenate it with the current frame image patch. However, this approach does not take the ego-motion of the lidar sensor into account. As the lidar moves over time, the range image patch with the same rows and columns can correspond to vastly different physical space.

To take sensor poses into consideration, instead of querying pixels of the last frame using the row and column indices, we should query neighbors using 3D points in the global coordinate (Fig. 3). However, as we do not know the ground truth range value for the pixel (i,j), we have to approximate the query by using a predicted range (e.g. using the left pixel range or the predicted value from the intra-frame model). Given pixel (i,j)'s laser shot angle (θ,α) and its estimated range $\hat{r}$, we get a point in the global coordinate, following Sec. 3. Then given the points from last frame in the global coordinate, we can directly query neighbors in the 3D space (using KDtrees to accelerate the query). Those neighboring points from last frame can then be projected to laser shot (i,j)'s spherical coordinate (to the points in the sensor coordinate at the time of the laser shot and then transform to the spherical coordinate), to obtain extra points as temporal contexts. [2] This is equivalent to assuming the points from the last frame are static, and we re-scan the scene at the sensor location at the time of the laser shot (i,j). To distinguish the points from the last and current frames, we augment the points with an extra time channel (with 1 indicating the last frame and 0 indicating the current frame).

Note that the reprojected points from the last frame do not directly correspond to the rows and columns of the current frame range image. Considering such input as a point cloud is convenient as we do not require any interpolation to turn the points to the image grid or any predefined neighborhood size for image cropping.

Inference. At inference time (for compression), we start from the top left patch of the range image to predict pixel I'_1 or $I'_{1,1}$ and store the residual. This process continues in a raster scanning order to predict pixels $I_{1,2},...,I_{1,W},I_{2,1},...,I_{i,j},...,I_{H,W}$. The residual map (deltas between the prediction and quantized values) of size $H \times W$ would be compressed by the entropy encoder. At decompression time, we run the prediction model in the same raster-scanning order, which takes input as already reconstructed pixels $\{I'_1,..,I'_{k-1}\}$, predicts the next pixel value $\hat{I}_k$ and then reconstruct the pixel from saved residual as $I'_k = \hat{I}_k + \delta_k$, where δ_k is the stored delta of pixel $k=(i-1)W+j$. This process can be parallelized by dividing the input range image into blocks and run the inference

[2] Strictly, even the pixels/points from the current frame need be reprojected to the sensor coordinate at the time of the shot (i,j). We have this reprojection in our intra-frame model but the impact is small as the sensor moves little between a few pixels.

in parallel for each block (discussed in the supplementary).

4.3. Entropy Encoding

After the predictive delta encoding, we get a residual map/array of the range image. An entropy encoder is used to leverage the sparsity pattern in the residual map to compress it. Given an accurate prediction model, most of the residuals would be zero. We adopt two methods to entropy encode the residuals. In practice, we select the entropy encoder with the highest compression rates depending on the quantization rates and the predictor.

The first method is to represent the residuals using a sparse representation, with the values of the nonzero residuals and their indices in the array, which can then be arithmetically encoded to further reduce its size. The second method is to represent the residuals using run-length encoding, which achieves better compression rates when the residuals are not very sparse, i.e., when quantization step is small. After obtaining the run-length representation, we use LZMA compressor to further reduce its size.

5. Experiments

In this section, we first introduce the datasets and the metrics in Sec. 5.1. Then we report compression results compared with strong baselines and prior art methods in Sec. 5.2 both quantitatively and qualitatively. We further evaluate the impact of compressed data to downstream perception tasks (3D detection of vehicles and pedestrians) in Sec. 5.3. Finally, we provide extensive analysis experiments to validate our design choices in Sec. 5.4.

5.1. Dataset and Metrics

Waymo Open Dataset (WOD) [26] WOD is the main dataset we experiment with, as it provides rich lidar calibration data and full sensor poses. WOD includes a total number of 1,150 sequences with 798 for training and 202 for validation. Each sequence lasts around 20 seconds with a sampling frequency of 10Hz. A 64-beam lidar is used, providing range images of 64 rows and 2,650 columns, with provided lidar calibration metadata (beam inclination angles). The range channel is cropped to 75m, and each raw range value is stored as a 32-bit float in default. We use the training set to train our deep model and evaluate on the validation set. Only the first return range images are used in our experiments.

SemanticKITTI [8] We also evaluate our method on SemanticKITTI (which enhances KITTI [13] with semantic labels) to compare with prior art methods OctSqueeze [16] and MuSCLE [9] (since they do not release code, we cannot compare with them on the WOD). We directly apply the WOD trained model on SemanticKITTI test split (sequence 11-21). However, as KITTI only released the point cloud data but not the the raw range images nor the sensor poses, we have to refer to the manual of the Velodyne lidar [2] used by KITTI to convert a point cloud to the spherical coordinate to get a *pseudo* range image with 64 rows and 2,088 columns. For our method, we compress the pseudo range images and do not additionally store the azimuth and elevation of the pixels, as their storage in actual Velodyne range images are negligible (elevations are known and azimuths can be compressed to less than 1Kb per frame [32]).

Metrics Following previous works [9, 14, 16], we use two geometric metrics to evaluate the reconstruction quality of the compressed point cloud data: point-to-point Chamfer distance and point-to-plane peak signal-to-noise ratio (PSNR). We report these metrics as a function of bitrates i.e., the average number of bits to store one lidar point.

The point-to-point Chamfer distance CD_{sym} measures the average point distances between two point clouds (smaller the better). For a given point cloud $P = \{p_i\}_{i=1,\dots N}$ and the reconstructed point cloud $\hat{P} = \{\hat{p}_j\}_{j=1,\dots M}$:

$$\text{CD}(P, \hat{P}) = \frac{1}{|P|} \sum_i \min_j \|p_i - \hat{p}_j\|_2 \tag{2}$$

$$\text{CD}_{sym}(P, \hat{P}) = \max\{\text{CD}(P, \hat{P}), \text{CD}(\hat{P}, P)\} \tag{3}$$

The second metric, the peak signal-to-noise ratio (PSNR) [28] (the larger the better), measures the ratio between the "resolution" of the point cloud r and the average point-to-plane error between the original point cloud P and the reconstructed point cloud $\hat{P}$:

$$\text{PSNR}(P, \hat{P}) = 10 \log_{10} \frac{r^2}{\max\{\text{MSE}(P, \hat{P}), \text{MSE}(\hat{P}, P)\}} \tag{4}$$

where $\text{MSE}(P, \hat{P}) = \frac{1}{|P|} \sum_i ((p_i - \hat{p}_i) \cdot n_i)^2$ is the point-to-plane distance, $\hat{p}_i$ is the closest point in $\hat{P}$ to p_i, $r = \max_{p_i \in P} \min_{j \neq i} \|p_i - p_j\|_2$ is the intrinsic resolution of the original point cloud. We estimate the normal n_i using Open3D [36] with $k = 12$ for k nearest neighbor.

5.2. Compression Results

In this section, we compare our methods with competitive baselines as well as prior art lidar data compression methods. We focus on compressing the range channel or the 3D coordinates of the points as it is the most studied attribute among the others (intensity, elongation) and some of the methods in comparison do not support compressing other attributes. See supplementary material for more results on compressing the other channels. We adjust the quantization precision of the range images to achieve different compression rates (bits per point) of our method.

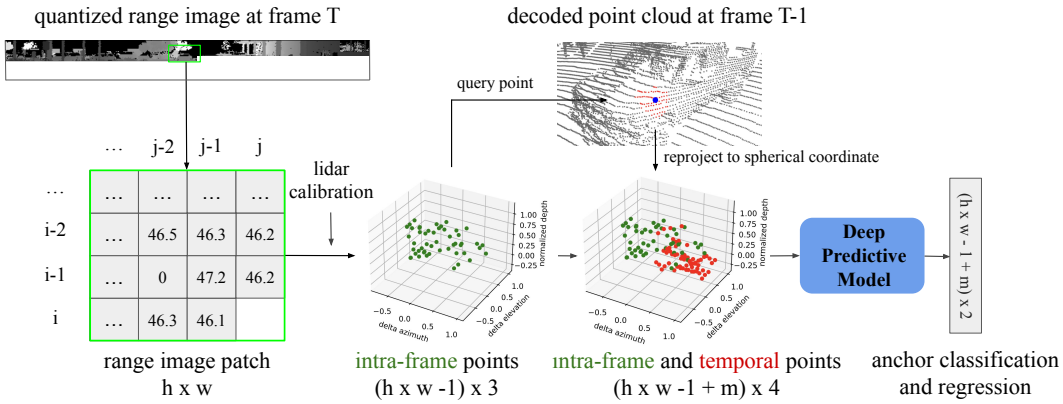

Figure 3. **The deep prediction model.** Given a range image patch from frame T with quantized attribute values (e.g. range), we lift pixels to the spherical coordinate with azimuth and elevation angles from lidar calibration. To leverage context points from the past frame T-1, a query point is generated to find neighbors among points at frame T-1. Those neighbor points are then projected to the spherical coordinate of the pixel to be predicted. Our predictor takes the union of the intra-frame and temporal context points and predicts the attribute of the pixel (i, j) with anchor classification and regression (with each input point as an anchor).

Baselines: **G-PCC** [14] is a point cloud compression method proposed by the MPEG, using octrees. **Draco** [1] is a popular point cloud compression algorithm based on Kdtrees proposed by Google. We also compare with two prior art deep model based methods [3]: **OctSqueeze** [16] is a octree-based method that uses a neural network to predict the next-level symbol of the octree; **MuSCLE** [9] further strengthens OctSqueeze by leveraging multi-sweep (temporal) data for the octree prediction. In terms of range image representation, we compare with **PNG** (intra-frame) as well as **HEVC** (a video compression standard) on top of PNG for temporal range image compression. For the PNG compression, the range is coded with 16 bits with a varying scaling factor to control the distortion/compression rate. We also compare with **Cluster** [27], a range image-based lidar data compression algorithm with a pipeline of segmentation, clustering, 3D-HEVC encoding and ground prediction. Besides, supplementary provides a further experiment comparing with an auto-encoder based method on range images (not included here due to its poor performance).

Implementation Details Our intra-frame prediction model, **RIDDLE**, takes in a context image patch of size 10×10 (the bottom right pixel is masked out) and uses a PointNet [22] like architecture for the prediction (without the T-Net structure, adapted the output to predict anchor classification and regression). The input to the network is a 3D point cloud in a spherical coordinate with azimuth, elevation relative to the bottom right pixel and the range relative to the mean range of valid context points. Our temporal model, **RIDDLE-T**, uses the same network architecture as the intra-frame one but takes in an extra 100 points from the last scan (projected to the spherical coordinate of the next pixel). Please see supplementary for more details.

Waymo Open Dataset Results We report the bitrate versus reconstruction quality metrics (PSNR, Chamfer distance) of competing methods on all frames from the sequences in the validation set of the Waymo Open Dataset. As shown in Fig. 4, our method significantly outperforms prior methods. At the same Chamfer distance around 0.005, our method reduces the bitrate by more than 65% compared to G-PCC (from 10.78 bpp to 3.65 bpp). At the bitrate of around 4, our method reduces the distortion (measured by Chamfer distance) by more than 85%. Our method also has a larger bitrate improvement over previous methods when the reconstruction quality is higher. This indicates our method has more advantage over baselines when the data quality requirement is higher.

SemanticKITTI Results Since prior art methods [9, 16] have not released the code or the compression model, we turn to the SemanticKITTI dataset to compare with them (we got the raw values of the curves reported in the MuSCLE [9] paper from the authors). We apply our model trained on the Waymo Open Dataset directly to the SemanticKITTI lidar point clouds (by creating pseudo range images).

As shown in Fig. 5, our method is more than 50% lower in bitrate (at around 4.3 bpp) with the same Chamfer distance at around 0.005 compared to all prior art methods, showing significant advantages. This strong lead attributes

[3]There is another deep net based work VoxelContextNet [23], yet as they did not release code nor the detailed definition of the evaluation metrics, we could not compare with them.

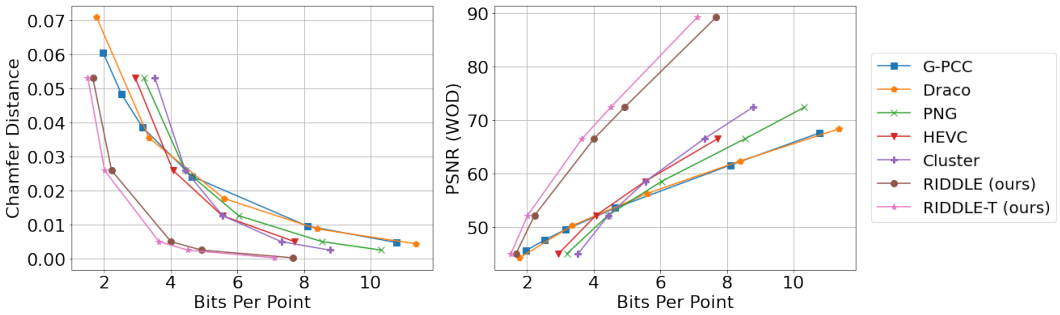

Figure 4. **Evaluation of the compression methods with geometric metrics on the *Waymo Open Dataset val set*.** *Left*: Chamfer distance v.s. bit per point (bbp); *Right*: PSNR v.s. bpp. At a certain bitrate, lower the Chamfer distance or higher the PSNR, better the reconstruction quality.

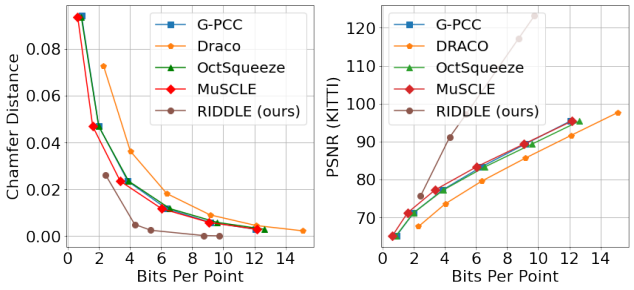

Figure 5. **Evaluation of the compression methods with geometric metrics on the *SemanticKITTI test set*.** We only present our intra-frame model here as the per pixel sensor pose is unavailable in SemanticKITTI.

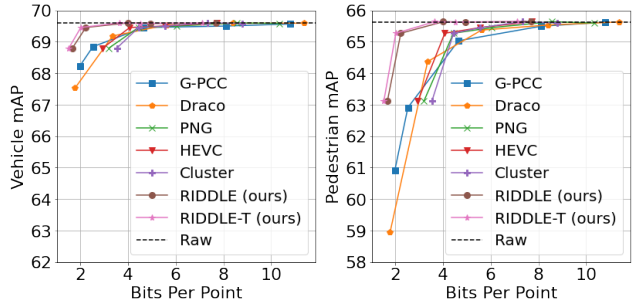

Figure 6. **Impact of lidar data compression to 3D object detection quality on the *Waymo Open Dataset val set*.** We train PointPillars [17] detectors using the raw point clouds (with no compression) from the WOD train set and evaluate them with the compressed point clouds (or point clouds from the compressed range images) on the WOD validation set.

to our choice of directly compressing the range images as well as the effective deep model.

Qualitative results. In Fig. 7, we show the reconstructed lidar point clouds from our method, Draco and G-PCC. We can see that the point cloud reconstructed from our method remarkably resembles the original point cloud in geometry even when the bitrate is ambitiously set very low, thanks to compressing directly on the range images to keep the point distribution pattern.

5.3. Impact to Downstream Perception Tasks

For applications like autonomous driving, we want to understand the impact of lidar data compression to downstream perception tasks such as 3D object detection. To understand such impact, we trained a widely used PointPillars detector [17] on uncompressed point clouds using the Waymo Open Dataset train set, for the vehicle class and pedestrian class respectively. Detection quality is measured by mean average precision (mAP).

As shown in Fig. 6, our method outperforms other competing baselines in maintaining the best mAP with the same bitrate. At the bitrate around 2, our method leads the second best method (G-PCC) by more than 1 point on vehicle detection and 3 points on pedestrian detection. We can also see that pedestrian detection is more sensitive to data distortion probably due to the smaller average object sizes compared to vehicles.

5.4. Analysis Experiments

In this section we ablate our deep model in terms of architecture choice, loss design and temporal context. In order to compare prediction quality independent from the entropy encoder, we use a prediction accuracy as the metrics for ablation studies. The prediciton accuracy (acc.) is defined as the percentage of zero deltas (i.e. perfect prediction under quantization) in the range image residual map, under a specific quantization precision (e.g. $\delta = 0.1\text{m}$ [4]). A prediction q for the quantized range value p' is counted as correct if $|q - p'| < \delta/2$. Supplementary provides more analysis related to entropy encoders and model latency.

[4]Note 0.1m is not that coarse as average point displacement after the quantization is only 2.5cm

model	acc. @0.1m
previous valid value	54.35
linear interpolation	54.64
12-layer CNN	64.62
PointNet (adpated)	**65.75**

Table 1. Effects of prediction models.

loss function	acc. @0.1m
MSE	59.83
MAE	61.64
multi-bin loss	59.66
anchor cls. + reg.	**65.75**

Table 2. Effects of loss functions.

temporal context	acc. @0.1m
none (intra-frame)	65.75
10×10 image	67.34
100 knn points	**69.23**

Table 3. Effects of temporal input.

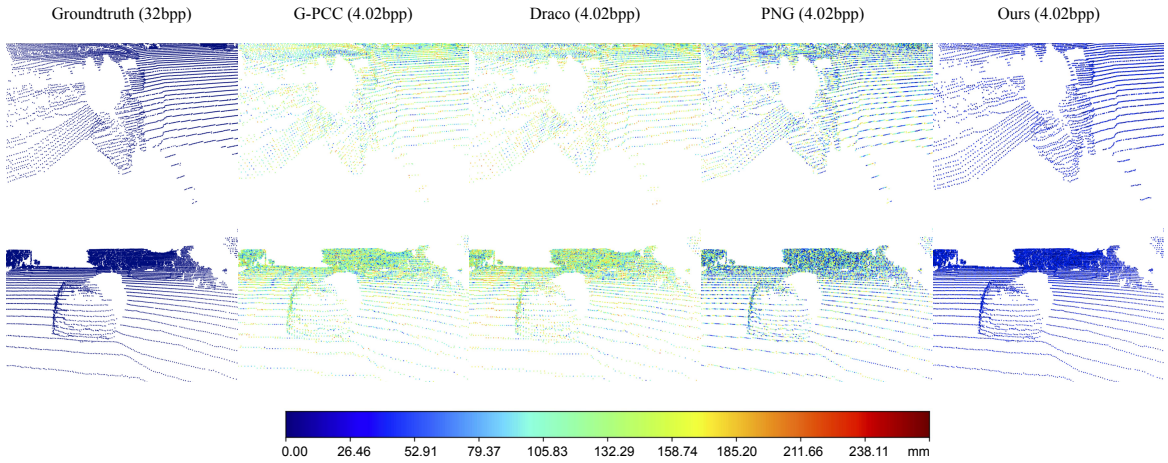

Figure 7. **Visualization of reconstructed point clouds, colored by per point Chamfer distance** (error bar colormap on the bottom). From left to right: raw, G-PCC, Draco, PNG and RIDDLE (ours). It is clear that our method, under the same bit per point, has mush less distortion. Best viewed in color with zoom in.

Effects of predictor choices. Table 1 compares several architecture choices. The simplest choice is to use the left valid pixel as the prediction to the current pixel: $\hat{I}_{i,j} = I'_{i,j-1}$. Another extension is to use linear interpolation of close-by pixels: $\hat{I}_{i,j} = I'_{i,j-1} + I'_{i-1,j} - I'_{i-1,j-1}$. Note that for both cases, first valid pixel is used in case the nearby one is an empty pixel. We see that deep models can significantly outperform linear models while the point-cloud-based architecture shows a stronger empirical result compared to ConvNet on the image representation.

Effects of loss functions. Table 2 compares several loss choices for our model supervision. With direct attribute prediction as a regression problem, we can see using the mean absolute error (MAE, L1 loss) is superior to using the mean squared error (MSE, L2 loss) as it is affected less by the large errors on the object boundaries. Turning the depth regression problem to a multi-bin classification and regression problem (with classification and intra-bin regression for each depth bin of size 1m) does not help much either as shown in the third row. Our proposed formulation (anchor classification with regression) leads to 4.11 points increase in prediction accuracy compared to the second best option of using mean absolute error.

Effects of temporal contexts. Table 3 shows the benefits of adding temporal contexts to the prediction model. We see that even the naive concatenation of the image patch of the last frame with the same rows and columns (second row) can already help. A more careful handling of the temporal points by considering sensor poses (as described in Sec. 4.2) leads to more gains of using the temporal data.

6. Conclusion

With improving lidar sensor resolution and growing data volume, how to efficiently store and transmit lidar data becomes a challenging problem in many 3D applications, such as autonomous driving and augmented reality. To address this challenge, we propose a novel lidar data compression algorithm named RIDDLE (Range Image Deep DeLta Encoding), which combines the succinctness of traditional delta encoding and the expressiveness of deep neural networks, with support of using temporal contexts. Experiments over the Waymo Open Dataset and KITTI show that compared to previous methods, the proposed approach yields significant improvement in the point cloud reconstruction quality and the downstream perception model performance, under the same compression rates.

References

[1] Draco. https://github.com/google/draco. Accessed: 2021-09-28. 2, 6

[2] Velodyne hdl-64e. https://gpsolution.oss-cn-beijing.aliyuncs.com/manual/LiDAR/MANUAL%2CUSERS%2CHDL-64E_S3.pdf. Accessed: 2021-10-04. 5

[3] Jae-Kyun Ahn, Kyu-Yul Lee, Jae-Young Sim, and Chang-Su Kim. Large-scale 3d point cloud compression using adaptive radial distance prediction in hybrid coordinate domains. *IEEE Journal of Selected Topics in Signal Processing*, 9(3):422–434, 2015. 2

[4] I. Armeni, A. Sax, A. R. Zamir, and S. Savarese. Joint 2D-3D-Semantic Data for Indoor Scene Understanding. *ArXiv e-prints*, Feb. 2017. 1

[5] Johannes Ballé, Valero Laparra, and Eero P Simoncelli. End-to-end optimized image compression. *arXiv preprint arXiv:1611.01704*, 2016. 2

[6] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. *arXiv preprint arXiv:1802.01436*, 2018. 2

[7] Peter van Beek. Image-based compression of lidar sensor data. *Electronic Imaging*, 2019(15):43–1, 2019. 2

[8] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Jurgen Gall. Semantickitti: A dataset for semantic scene understanding of lidar sequences. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9297–9307, 2019. 5

[9] Sourav Biswas, Jerry Liu, Kelvin Wong, Shenlong Wang, and Raquel Urtasun. Muscle: Multi sweep compression of lidar using deep entropy models. *arXiv preprint arXiv:2011.07590*, 2020. 1, 2, 5, 6

[10] Mario Botsch, Andreas Wiratanaya, and Leif Kobbelt. Efficient high quality rendering of point sampled geometry. *Rendering Techniques*, 2002:13th, 2002. 2

[11] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *CVPR*, 2017. 1

[12] Olivier Devillers and P-M Gandoin. Geometric compression for interactive transmission. In *Proceedings Visualization 2000. VIS 2000 (Cat. No. 00CH37145)*, pages 319–326. IEEE, 2000. 2

[13] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013. 2, 5

[14] D Graziosi, O Nakagami, S Kuma, A Zaghetto, T Suzuki, and A Tabatabai. An overview of ongoing point cloud compression standardization activities: video-based (v-pcc) and geometry-based (g-pcc). *APSIPA Transactions on Signal and Information Processing*, 9, 2020. 2, 5, 6

[15] Hamidreza Houshiar and Andreas Nüchter. 3d point cloud compression using conventional image compression for efficient data transmission. In *2015 XXV International Conference on Information, Communication and Automation Technologies (ICAT)*, pages 1–8. IEEE, 2015. 2

[16] Lila Huang, Shenlong Wang, Kelvin Wong, Jerry Liu, and Raquel Urtasun. Octsqueeze: Octree-structured entropy model for lidar compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1313–1323, 2020. 1, 2, 5, 6

[17] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *CVPR*, 2019. 7

[18] Siwei Ma, Xinfeng Zhang, Chuanmin Jia, Zhenghui Zhao, Shiqi Wang, and Shanshe Wang. Image and video compression with neural networks: A review. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(6):1683–1698, 2019. 2

[19] Fabian Mentzer, Eirikur Agustsson, Michael Tschannen, Radu Timofte, and Luc Van Gool. Practical full resolution learned lossless image compression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10629–10638, 2019. 2

[20] Fabrizio Nenci, Luciano Spinello, and Cyrill Stachniss. Effective compression of range data streams for remote robot operations using h.264. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3794–3799, 2014. 2

[21] Aaron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. Conditional image generation with pixelcnn decoders. *arXiv preprint arXiv:1606.05328*, 2016. 2, 4

[22] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017. 4, 6

[23] Zizheng Que, Guo Lu, and Dong Xu. Voxelcontext-net: An octree based framework for point cloud compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6042–6051, 2021. 1, 2, 6

[24] Ruwen Schnabel and Reinhard Klein. Octree-based point-cloud compression. In *PBG@ SIGGRAPH*, pages 111–120, 2006. 2

[25] Shuran Song, Samuel P Lichtenberg, and Jianxiong Xiao. Sun rgb-d: A rgb-d scene understanding benchmark suite. In *CVPR*, 2015. 1

[26] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2446–2454, 2020. 2, 3, 5

[27] Xuebin Sun, Han Ma, Yuxiang Sun, and Ming Liu. A novel point cloud compression algorithm based on clustering. *IEEE Robotics and Automation Letters*, 4(2):2132–2139, 2019. 6

[28] Dong Tian, Hideaki Ochimizu, Chen Feng, Robert Cohen, and Anthony Vetro. Geometric distortion metrics for point cloud compression. In *2017 IEEE International Conference*

on Image Processing (ICIP), pages 3460–3464. IEEE, 2017. 5

[29] George Toderici, Damien Vincent, Nick Johnston, Sung Jin Hwang, David Minnen, Joel Shor, and Michele Covell. Full resolution image compression with recurrent neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5306–5314, 2017. 2

[30] James Townsend, Thomas Bird, Julius Kunze, and David Barber. Hilloc: Lossless image compression with hierarchical latent variable models. *arXiv preprint arXiv:1912.09953*, 2019. 2

[31] Chenxi Tu, Eijiro Takeuchi, Alexander Carballo, and Kazuya Takeda. Point cloud compression for 3d lidar sensor using recurrent neural network with residual blocks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3274–3280. IEEE, 2019. 1, 2

[32] Chenxi Tu, Eijiro Takeuchi, Chiyomi Miyajima, and Kazuya Takeda. Compressing continuous point cloud data using image compression methods. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1712–1719. IEEE, 2016. 3, 5

[33] Aaron Van Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *International Conference on Machine Learning*, pages 1747–1756. PMLR, 2016. 1, 2

[34] Louis Wiesmann, Andres Milioto, Xieyuanli Chen, Cyrill Stachniss, and Jens Behley. Deep compression for dense point cloud maps. *IEEE Robotics and Automation Letters*, 6(2):2060–2067, 2021. 2

[35] Wei Yan, Shan Liu, Thomas H Li, Zhu Li, Ge Li, et al. Deep autoencoder-based lossy geometry compression for point clouds. *arXiv preprint arXiv:1905.03691*, 2019. 2

[36] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3d: A modern library for 3d data processing. *arXiv preprint arXiv:1801.09847*, 2018. 5