Density-preserving Deep Point Cloud Compression (Supplementary Material)

Yun He[1*] Xinlin Ren[1*] Danhang Tang[2] Yinda Zhang[2] Xiangyang Xue[1] Yanwei Fu[1]
[1] Fudan University [2] Google

In this document, we provide additional implementation details, ablation studies and qualitative results. Limitation and potential ethic concerns are also discussed.

1. Implementation Details

Additional details about hyperparameter settings, detailed network architecture, and diagrams of baselines used in ablation study are elucidate in this section. We also formulate the reconstruction metrics used in experiments.

1.1. Hyperparameters

In the experiments, we choose the number of stages $S = 3$ and the downsampling factor $f_s \in \{1/2, 1/3\}$. We set dimension $d = 8$ for all three embeddings extracted by the encoder, and maximum upsampling factor $U = 8$ in the decoder. For distortion loss, we set the weight of density loss $\alpha = 1e-4$ and cardinality loss $\beta = 5e-7$. Where in density loss, the coefficient $\gamma = 5$. And the weight μ for density metric is setted as 1. For normal compression, we additionally add a L2 loss between the reconstructed normals and ground truth , and its weight is $1e-2$. To obtain the rate-distortion trade-off curves, we vary the coefficient of rate loss λ and downsampling factor f_s. Moreover, in the adaptive scale upsampling block, we use an icosahedron to sample uniformly on a unit sphere to get $M = 43$ candidate directions, which include 42 vertices of the icosahedron and 1 origin, following [12].

Our model is implemented with Pytorch [8] and CompressAI [1], trained on a NVIDIA TITAN X GPU for 50 epochs. We use the Adam optimizer [6] with a initial learning rate of 1e-3 and a decay factor of 0.5 every 15 epochs.

1.2. Detailed Network Architecture

The detailed architectures of our encoder and decoder are shown in Fig 1 and Fig 2 separately. At stage s of the encoder, for each point $p \in \mathcal{P}_{s+1}$, we first extract local position embedding $\mathbf{F}^{\mathcal{P}}$ and density embedding $\mathbf{F}^{\mathcal{D}}$ to capture the local geometry and density information of current stage. And ancestor embedding $\mathbf{F}^{\mathcal{A}}$ is also utilized to aggregate features from previous stages by applying point transformer layer [14], based on the collapsed points set $\mathcal{C}(p)$. However, the cardinality of each point's collapsed set may be different. To achieve parallel process, we first find the k-nearest neighbor $\mathcal{K}(p)$ in $\mathcal{P}_s$ for each downsampled point p, and then apply a mask when using attention for feature aggregation. Specifically, the mask is defined as below:

$$w_k = \begin{cases} MLPs(\mathbf{F}(p_k)), & \text{if } p_k \in \mathcal{K}(p) \text{ and } p_k \in \mathcal{C}(p) \\ 0, & \text{else} \end{cases} \tag{1}$$

where w_k and $\mathbf{F}(p_k)$ are the weight and feature of p_k. We set $k = 20$, which is much larger than $1/f_s$, so the collapsed set $\mathcal{C}(p)$ is guaranteed to be the subset of $\mathcal{K}(p)$.

And in the decoder, we apply sub-point convolution to construct the scale-adaptive upsampling block, which promotes the recovering of local geometry patterns and density.

1.3. Reconstruction Metrics

We use the symmetric point-to-point Chamfer Distance and point-to-plane PSNR to evaluate the geometry accuracy of reconstructed point clouds. And now we list their mathematical formulas.

Given ground truth $\mathcal{P}_s$ and reconstructed point cloud $\hat{\mathcal{P}}_s$, the calculation of symmetric point-to-point Chamfer Distance is as follow:

$$CD(\mathcal{P}_s, \hat{\mathcal{P}}_s) = \frac{1}{|\mathcal{P}_s|} \sum_{p \in \mathcal{P}_s} \min_{\hat{p} \in \hat{\mathcal{P}}_s} \|p - \hat{p}\|_2^2 + \frac{1}{|\hat{\mathcal{P}}_s|} \sum_{\hat{p} \in \hat{\mathcal{P}}_s} \min_{p \in \mathcal{P}_s} \|\hat{p} - p\|_2^2 \tag{2}$$

Following [2], we calculate the symmetric point-to-plane PSNR as:

$$PSNR(\mathcal{P}_s, \hat{\mathcal{P}}_s) = 10 \log_{10} \frac{3\sigma^2}{\max\{MSE(\mathcal{P}_s, \hat{\mathcal{P}}_s), MSE(\hat{\mathcal{P}}_s, \mathcal{P}_s)\}} \tag{3}$$

where σ is the peak constant value, represented by the maximum nearest neighbor distance in the whole dataset [2].

*indicates equal contribution.

Yun He, Xinlin Ren and Xiangyang Xue are with the School of Computer Science, Fudan University.

Yanwei Fu is with the School of Data Science, Fudan University.

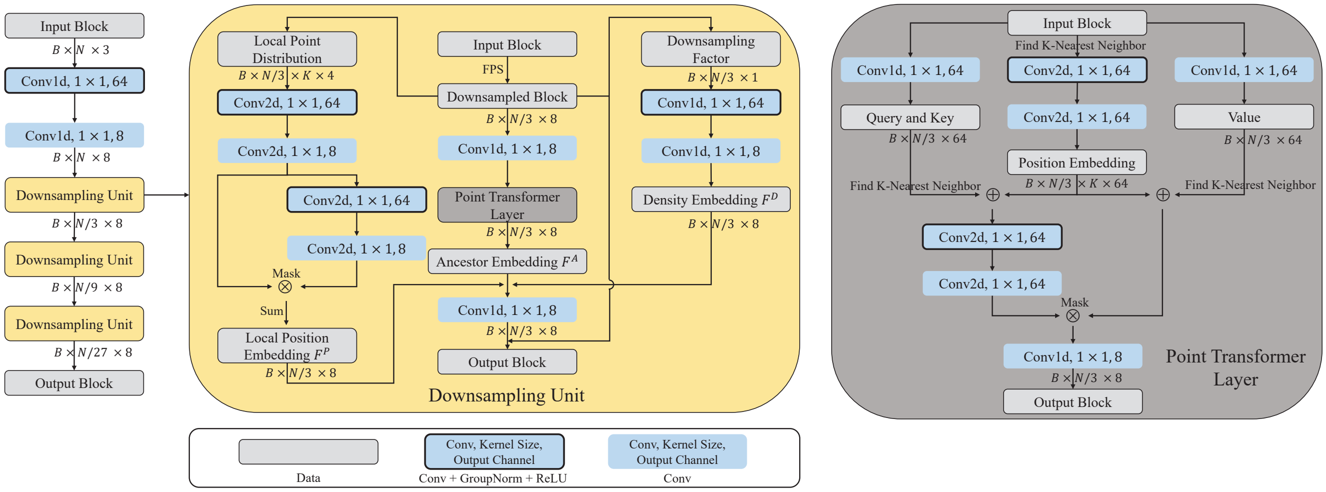

Figure 1. The detailed architecture of our encoder. The text below each box indicates the feature dimension: batchsize × points number × (k-nearest neighbor) × channel. And the stride of all convolution layers is fixed to 1.

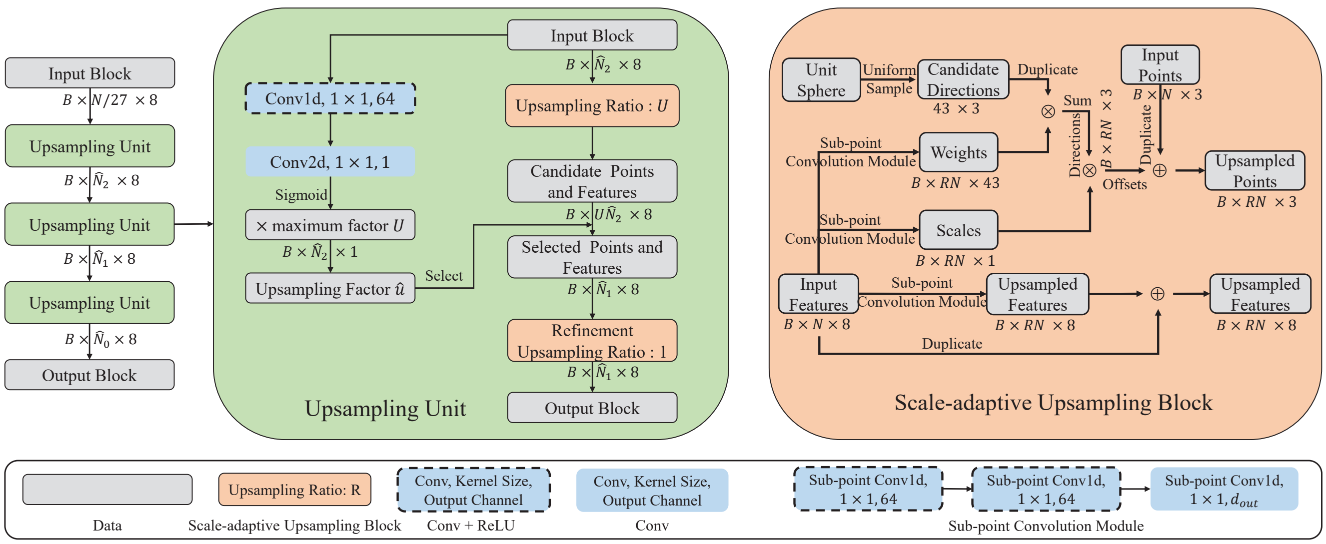

Figure 2. The detailed architecture of our decoder. The text below each box indicates the feature dimension: batchsize × points number × (k-nearest neighbor) × channel, where $\hat{N}_2 \approx N/9, \hat{N}_1 \approx N/3, \hat{N}_0 \approx N$. And the stride of all convolution layers is fixed to 1.

$MSE(\mathcal{P}_s, \hat{\mathcal{P}}_s) = \frac{1}{|\mathcal{P}_s|}\sum_{p\in\mathcal{P}_s}((p-\hat{p})\cdot\hat{n})^2$, where $\hat{p}$ is p's nearest neighbor in $\hat{\mathcal{P}}_s$, and $\hat{n}$ is the normal of $\hat{p}$. For each $p \in \mathcal{P}_s$, we estimate its normal using [15]. And for each $\hat{p} \in \hat{\mathcal{P}}_s$, we use the normal of its nearest neighbor in $\mathcal{P}_s$ as $\hat{n}$.

While Chamfer Distance and PSNR can only be used for position compression, we apply F1 score to measure the quality of both reconstructed locations and normals during normal compression, based on [2].

$$F_1(\mathcal{P}_s, \hat{\mathcal{P}}_s) = \frac{2TP}{2TP + FP + FN} \tag{4}$$

where TP (true positives) represent those reconstructed points $(\hat{p}, \hat{n}) \in \hat{\mathcal{P}}_s$ which have a corresponding ground truth point $(p, n) \in \mathcal{P}_s$ that satisfies $||p-\hat{p}||_2 \le \tau_p$ and $||n-\hat{n}||_2 \le \tau_n$; FP (false positives) indicate the rest reconstructed points; and FN (false negatives) are those ground truth points which do not have a corresponding TP. For SemanticKITTI, we set $\tau_p = 0.5, \tau_n = 0.5$; and for ShapeNet, we set $\tau_p = 0.05, \tau_n = 0.2$.

1.4. Baselines in Ablation Study

In the Table 2 of main paper, we validate the effectiveness of each component in our method. To achieve so, we first build a baseline model, which is composed of a point transformer encoder [14], entropy encoder and multi-branch MLPs decoder [13]. And we utilize a fixed upsampling factor $1/f_s$ for this baseline. Then we add the following components incrementally: dynamic upsampling factor $\hat{u}$, local position embedding $\mathbf{F}^{\mathcal{P}}$, density embedding $\mathbf{F}^{\mathcal{D}}$, scale-adaptive upsampling block, sub-point convolution and upsampling refinement layer. Here we draw the detailed structures of each model, as shown in Fig 3. Note that for all these models, we adopt the same pipeline, and only enable our contributing component once a time.

2. Additional Ablation Studies

In this section, we conduct some more ablation experiments to validate our choices of downsampling methods and loss functions. And all these experiments are conducted on SemanticKITTI with fixed bpp 2.1, the same as the main paper.

2.1. Downsampling Methods

At stage s of the encoder, we use FPS to get the downsampled point cloud $\mathcal{P}_{s+1}$, which expected to have a good coverage of the input $\mathcal{P}_s$. Besides FPS, there are also two common downsampling methods: random downsampling (RD) and grid downsampling (GD) [11]. And we replace FPS with these two downsampling methods in turn, as shown in Table 1. It is obvious that FPS can significantly improve the accuracy of reconstruction because it has better coverage, both in terms of geometry and local density.

Downsampling Methods	CD (10^{-2}) ↓	PSNR ↑	DM ↓
RD	1.29	41.17	3.08
GD	0.62	42.86	2.31
FPS	**0.36**	**44.03**	**1.98**

Table 1. The effectiveness of different downsampling methods. It is clear that FPS delivers the best performance.

2.2. Loss Functions

In our framework, we adopt the standard rate-distortion loss function for training. And the symmetric point-to-point Chamfer Distance D_{cha} is used as the distortion loss D, while the estimated bits number is used as the rate loss R. In addition to these two loss functions, we also extend the distortion loss by designing the density loss D_{den} and cardinality loss D_{card} to facilitate the recovery of local density. For validating the new loss functions D_{den} and D_{card}, we remove them degressively, as shown in Table 2.

As cardinality loss D_{card} is removed, all metrics drop slightly. However, once the constrain of local density is absent, the reconstruction quality will drop sharply, indicating the effectiveness of our designed density loss.

Loss Functions	CD (10^{-2}) ↓	PSNR ↑	DM ↓
Full Loss Functions	**0.36**	**44.03**	**1.98**
-D_{card}	0.47	43.62	2.15
-D_{den}	1.45	40.74	3.59

Table 2. The effectiveness of loss functions. Each row a loss function is romoved based on the top of previous row.

3. Additional Qualitative Results

In this section, we show more qualitative results on position compression, normal compression and downstream tasks, which clearly indicate that our density-preserving compression approach achieves the best performance. Specifically, in Fig 4, we show more qualitative position compression results on SemanticKITTI and ShapeNet. In Fig 5, we visualize the normal compression results by employing Poisson reconstruction [5] on decompressed points and normals. In Fig 6 and Fig 7, we display the qualitative results of two downstream tasks: surface reconstruction and semantic segmentation.

4. Limitation Discussion

Although our density-preserving deep point cloud compression framework is effective, it also has some limitations. For example: 1) The maximum upsampling factor U is predefined before decoding, thus the actual upsampling factor $\hat{u}$ is expected to be less than or equal to U. However, the assumption may be broken in some cases, especially when the local area is extremely dense, then our method may not be able to recover the local density precisely. 2) As we divide the point clouds into small blocks, each block may contain various number of points, so they are not easy to perfectly parallelized. 3) Other hyperparameters such as the weight of loss, the dimension of embedding, etc may be adaptively adjusted on different datasets. Moreover, we show some failure cases on extremely sparse point clouds in Fig 8. As we assume that there exists some data redundancy in the local areas of point clouds, so we can compress it while achieving tolerable distortion. However, this assumption may not hold when the point cloud is very sparse, and even the downsampled point cloud cannot describe the underlying geometry any more, hence is hard for reconstruction.

At last, we discuss the possible ethical issues. In general, since our point cloud compression algorithm is agnostic to the contents of point clouds, the responsibility of handling ethical issues belongs to the point cloud creator. That being said, as compressed point clouds may be intercepted by hackers during network transmission, which may result in data leakage, common encryption algorithms can be applied on the bottleneck point clouds and features to protect user privacy.

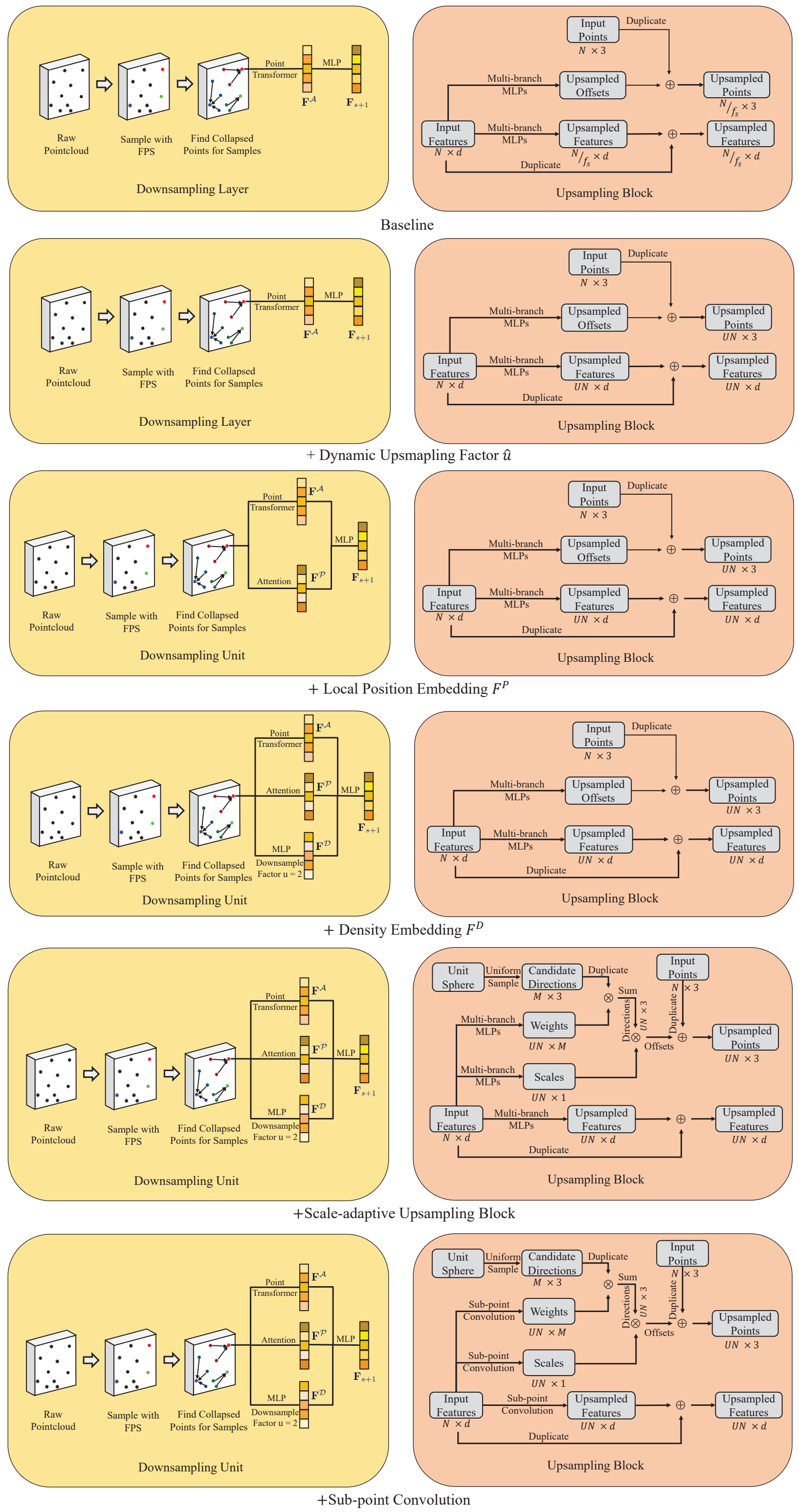

Figure 3. The detailed structures in ablation study (Table 2 of the main paper). Left: alternatives for the downsampling block in the encoder; right: alternatives for the scale-adaptive upsampling block in the decoder. For dynamic upsampling factor, we first generate U items and then select the first $\hat{u}$ points and features. While all these baselines do not have the refinement layer in the decoder, our full model adds it based on the "+Sub-point Convolution" model.

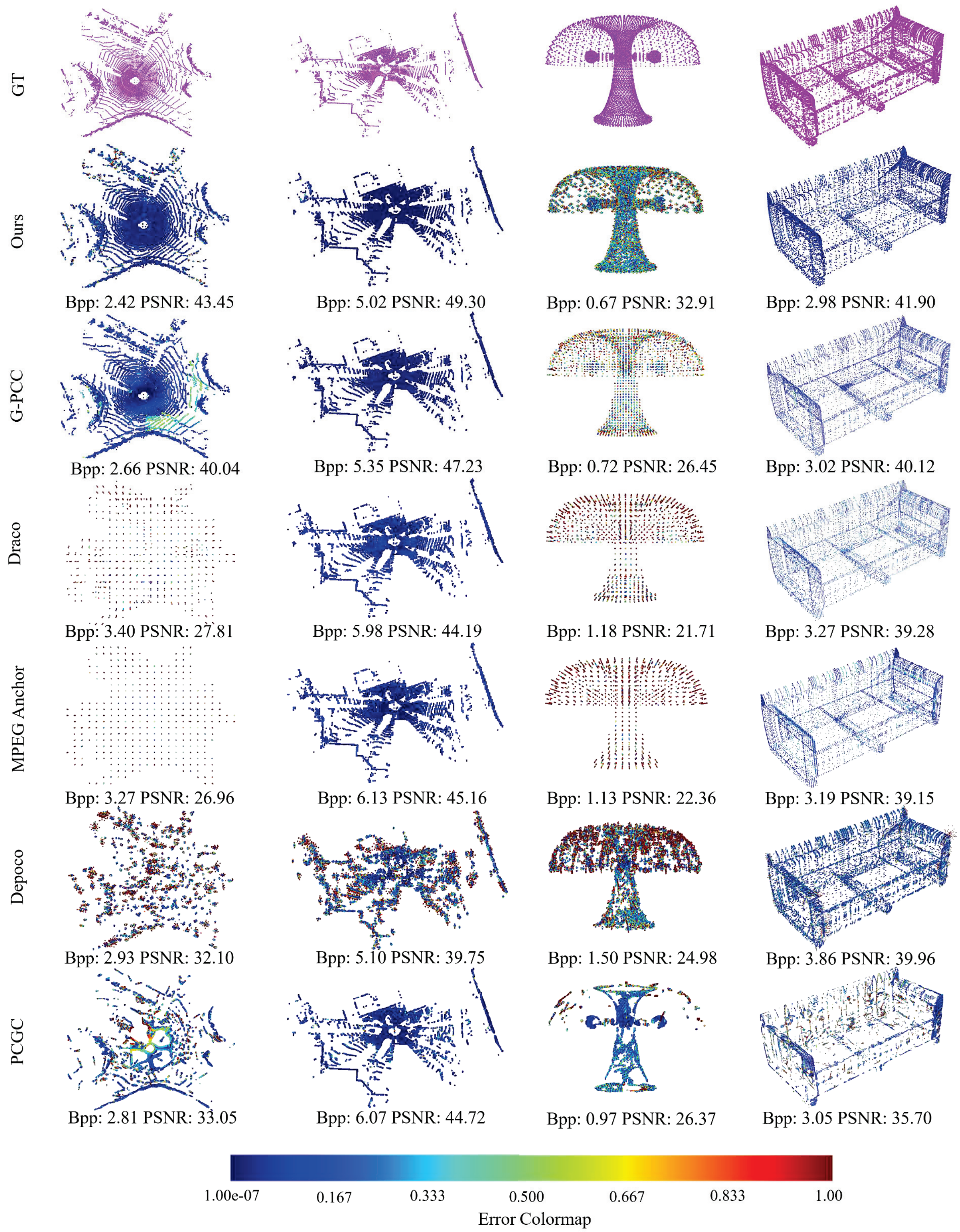

Figure 4. More qualitative results on SemanticKITTI (the first two coloums) and ShapeNet (the last two coloums). From top to bottom: Ground Truth, Ours, G-PCC [4], Draco [3], MPEG Anchor [7], Depeco [11] and PCGC [10]. We utilize the distance between each point in decompressed point clouds and its nearest neighbor in ground truth as the error. And the Bpp and PSNR metrics are averaged by each block of the full point clouds.

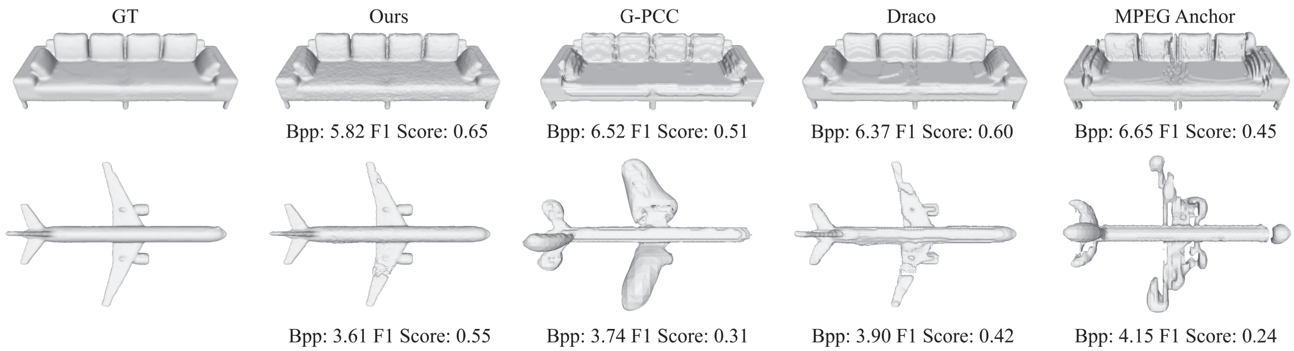

Figure 5. Qualitative results of normal compression. We apply Poisson reconstruction [5] to generate the mesh based on decompressed points and normals. And the Bpp and PSNR metrics are averaged by each block of the full point clouds.

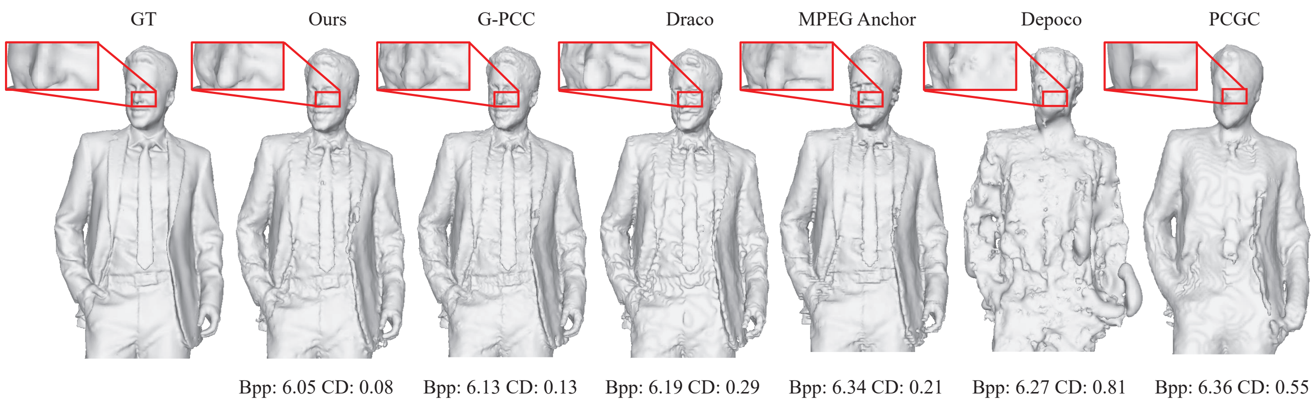

Figure 6. Qualitative results on the surface reconstruction downstream task, where CD represents the symmetric point-to-plane Chamfer Distance [9] on each full model. It is clear that the mesh reconstructed from our decompressed point cloud contains better surface details and more accurate geometry than others, especially on the face.

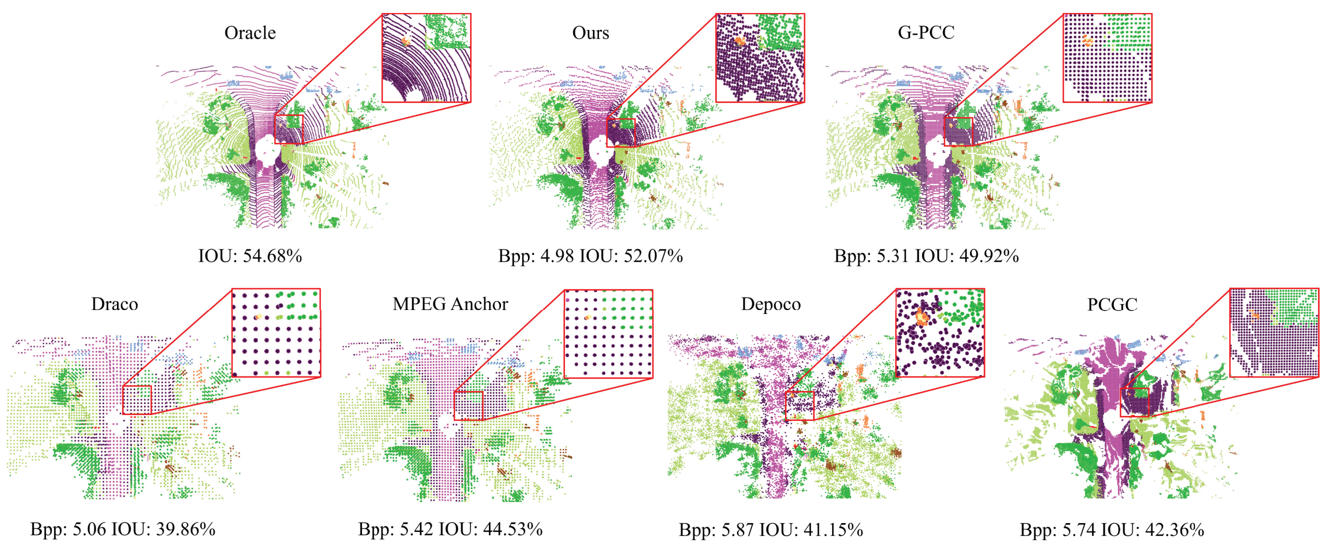

Figure 7. Qualitative results on the semantic segmentation downstream task, where IOU denotes the intersection-over-union metric on each scan. It is shown that preserving local density not only recovers more accurate geometry, but also benefits downstream task.

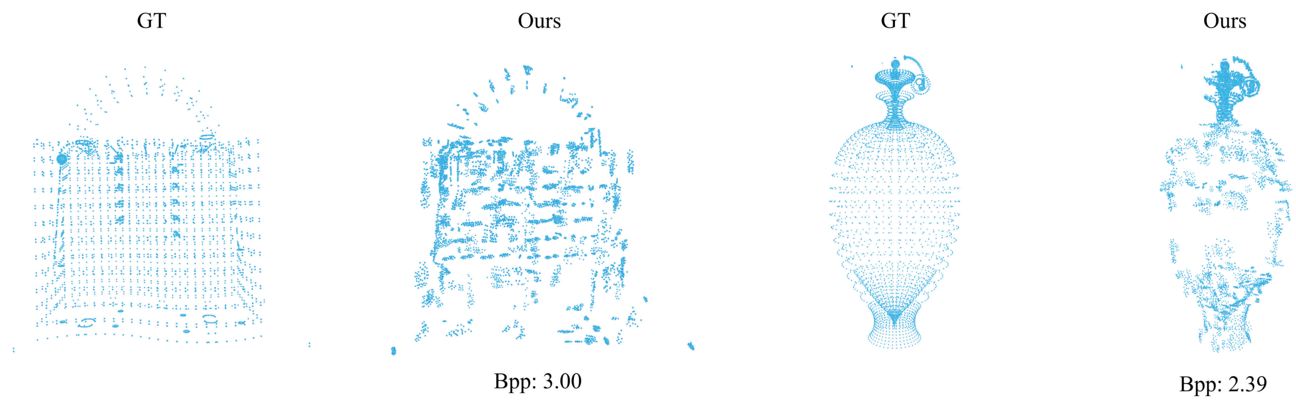

Figure 8. Failure cases on extremely sparse point clouds.

References

[1] Jean Bégaint, Fabien Racapé, Simon Feltman, and Akshay Pushparaja. Compressai: a pytorch library and evaluation platform for end-to-end compression research. *arXiv preprint arXiv:2011.03029*, 2020. 1

[2] 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

[3] Frank Galligan, Michael Hemmer, Ondrej Stava, Fan Zhang, and Jamieson Brettle. Google/draco: a library for compressing and decompressing 3d geometric meshes and point clouds. https://github.com/google/draco, 2018. 5

[4] 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. 5

[5] Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)*, 32(3):1–13, 2013. 3, 6

[6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 1

[7] Rufael Mekuria, Kees Blom, and Pablo Cesar. Design, implementation, and evaluation of a point cloud codec for tele-immersive video. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(4):828–842, 2016. 5

[8] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019. 1

[9] Danhang Tang, Saurabh Singh, Philip A Chou, Christian Hane, Mingsong Dou, Sean Fanello, Jonathan Taylor, Philip Davidson, Onur G Guleryuz, Yinda Zhang, et al. Deep implicit volume compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1293–1303, 2020. 6

[10] Jianqiang Wang, Hao Zhu, Haojie Liu, and Zhan Ma. Lossy point cloud geometry compression via end-to-end learning. *IEEE Transactions on Circuits and Systems for Video Technology*, 2021. 5

[11] 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. 3, 5

[12] Jianxiong Xiao, Tian Fang, Peng Zhao, Maxime Lhuillier, and Long Quan. Image-based street-side city modeling. In *ACM SIGGRAPH Asia 2009 papers*, pages 1–12. 2009. 1

[13] Lequan Yu, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. Pu-net: Point cloud upsampling network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2790–2799, 2018. 2

[14] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip HS Torr, and Vladlen Koltun. Point transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16259–16268, 2021. 1, 2

[15] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3d: A modern library for 3d data processing. *arXiv preprint arXiv:1801.09847*, 2018. 2