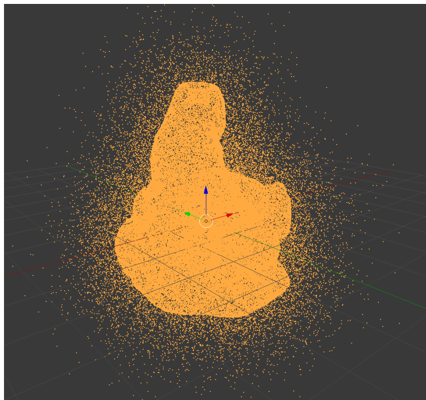
Processing of 3D Scans using Machine Learning

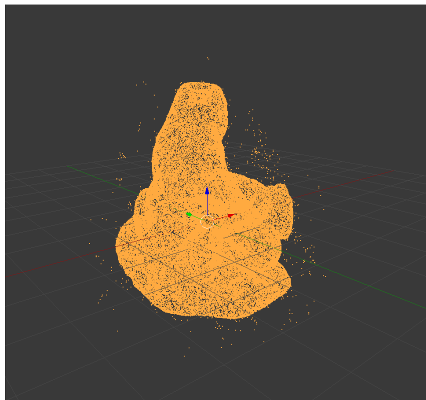
Lukáš Gajdošech

06.07.2020 - 20.07.2020

Removing Outliers - their data



(a) original



(b) processed

Idea

amount of outliers and noise. We assume the following point cloud formation model:

$$\mathbb{P}' = \{p'_i\} = \{p_i + n_i\}_{p_i \in \mathbb{P}} \cup \{o_j\}_{o_j \in \mathbb{O}}, \quad (1)$$

where $\mathbb{P}'$ is the observed noisy point cloud, $\mathbb{P}$ are perfect surface samples (i.e., $p_i \in \mathcal{S}$ lying on the scanned surface $\mathcal{S}$), n_i is additive noise, and $\mathbb{O}$ is the set of outlier points. We do not make any assumptions about the noise model n or the outlier model $\mathbb{O}$. The goal of our work is to take the low-quality point cloud $\mathbb{P}'$ as input, and output a higher quality point cloud closer to $\mathbb{P}$, that is better suited for further processing. We refer to this process as *cleaning*. We split the cleaning into two steps: first we remove outliers, followed by an esimation of per-point displacement vectors that denoise the remaining points:

$$\tilde{\mathbb{P}} = \{p'_i + d_i\}_{p'_i \in \mathbb{P}' \setminus \tilde{\mathbb{O}}}, \quad (2)$$

where $\tilde{\mathbb{P}}$ is the output point cloud, d are the displacement vectors and $\tilde{\mathbb{O}}$ the outliers estimated by our method. We first discuss our design choices regarding the desirable properties of the resulting point cloud and then how we achieve them.

(a) formulation

added noise and outliers. We then proceed in two stages: first, we train a non-linear function g that removes outliers:

$$\tilde{o}_i = g(\mathbb{P}'_i),$$

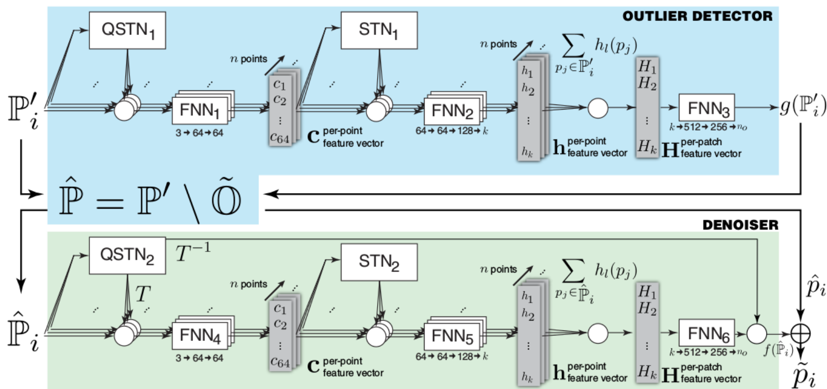
where $\tilde{o}_i$ is the estimated probability that point p'_i is an outlier. We add a point to the set of estimated outliers $\tilde{\mathbb{O}}$ if $\tilde{o}_i > 0.5$. After removing the outliers, we obtain the point cloud $\hat{\mathbb{P}} = \mathbb{P}' \setminus \tilde{\mathbb{O}}$. We proceed by defining a function f that estimates displacements for these remaining points to move them closer to the unknown surface:

$$d_i = f(\hat{\mathbb{P}}_i).$$

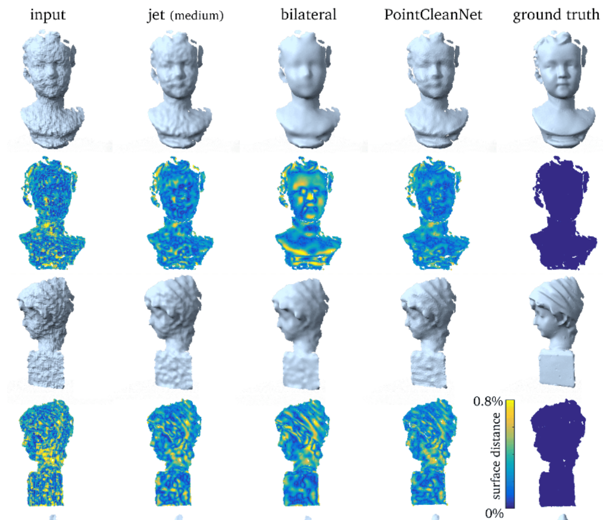
The final denoised points are obtained by adding the estimated displacements to the remaining noisy points: $\tilde{p}_i = \hat{p}_i + d_i$. Both f and g are modeled as deep neural networks with a PCPNet-based ar-

(b) stages

Architecture



Paper Results I - https://wang-ps.github.io/denoising.html



Paper Results II - https://www.blensor.org/

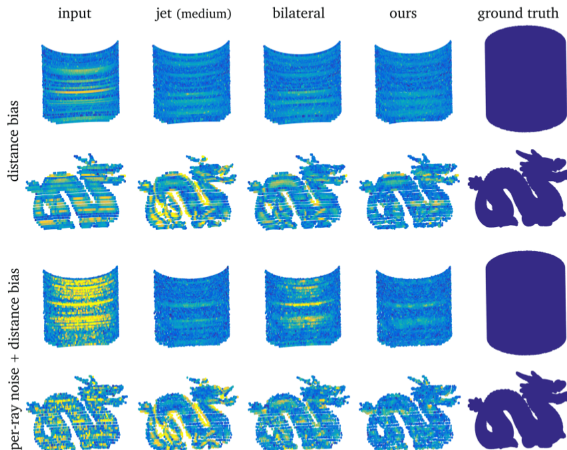
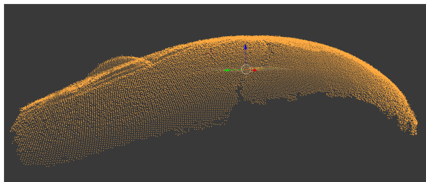
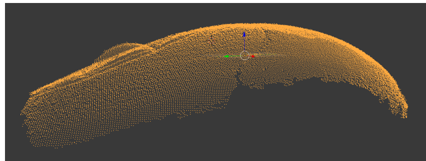


Figure 17: *Qualitative comparison with state-of-the-art methods on the Velodyne dataset. We display the normalized distance to the ground truth surface. The two top rows are evaluated on a dataset with only distance bias as noise and the two bottom rows with added per-ray noise. The simulated scanner noise has a high spatial correlation along the horizontal scan-lines, and lower correlation vertically across scan-lines. In this setting, jet fitting introduces significant error in detailed surface regions, while bilateral denoising has high residual error in the examples that have both noise types.* POINTCLEANNET *successfully learns the noise model, resulting in lower residual error.*

Denoising - our data



(a) original

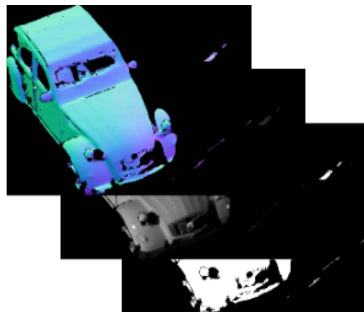


(b) processed

COGS Converter

COGS file

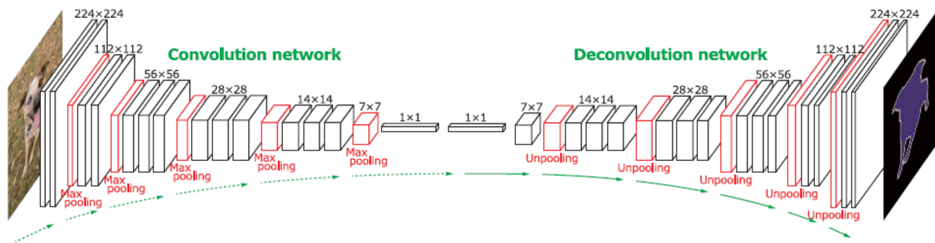
Converter



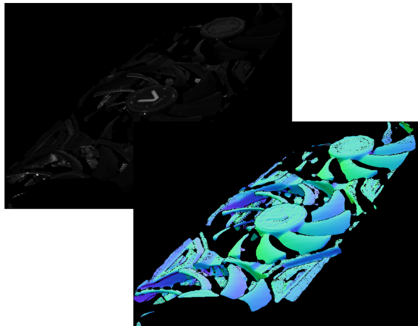
COGS Python

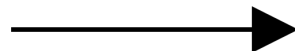
- Implement a Python wrapper for our C++ COGS library.
- Allows us to make an end-to-end system, taking a .cogs PC as input and directly returning its processed version.

AutoEncoder architecture



Example





Benchmark

GPU	API	backend	train time	inference time
RTX470	OpenCL	Plaid-ML	?	?
RTX470	OpenCL	TF (ROCm)	?	?
Vega64	OpenCL	Plaid-ML	?	3.2s
Vega64	OpenCL	TF (ROCm)	?	10.4s
RTX 2060	OpenCL	Plaid-ML	?	?
RTX 2060	CUDA	TensorFlow	?	?
RTX 2080	OpenCL	Plaid-ML	?	?
RTX 2080	CUDA	TensorFlow	?	?