Learning Local Displacements for Point Cloud Completion

Yida Wang\textsuperscript{1}, David Joseph Tan\textsuperscript{2}, Nassir Navab\textsuperscript{1}, Federico Tombari\textsuperscript{1,2}
\textsuperscript{1}Technische Universität München \textsuperscript{2}Google Inc.

Abstract

We propose a novel approach aimed at object and semantic scene completion from a partial scan represented as a 3D point cloud. Our architecture relies on three novel layers that are used successively within an encoder-decoder structure and specifically developed for the task at hand. The first one carries out feature extraction by matching the point features to a set of pre-trained local descriptors. Then, to avoid losing individual descriptors as part of standard operations such as max-pooling, we propose an alternative neighbor-pooling operation that relies on adopting the feature vectors with the highest activations. Finally, up-sampling in the decoder modifies our feature extraction in order to increase the output dimension. While this model is already able to achieve competitive results with the state of the art, we further propose a way to increase the versatility of our approach to process point clouds. To this aim, we introduce a second model that assembles our layers within a transformer architecture. We evaluate both architectures on object and indoor scene completion tasks, achieving state-of-the-art performance.

1. Introduction

Understanding the entire 3D space is essential for both humans and machines to understand how to safely navigate an environment or how to interact with the objects around them. However, when we capture the 3D structure of an object or scene from a certain viewpoint, a large portion of the whole geometry is typically missing due to self-occlusion and/or occlusion from its surrounding. To solve this problem, geometric completion of scenes \cite{2, 27, 32} and objects \cite{16, 20, 39, 44, 45} has emerged as a task that takes on a 2.5D/3D observation and fills out the occluded regions, as illustrated in Fig. 1.

There are multiple ways to represent 3D shapes. Point cloud \cite{3, 6}, volumetric grid \cite{8, 27}, mesh \cite{11} and implicit surfaces \cite{18, 21, 40} are among the most common data formats. These representations are used for most 3D-related computer vision tasks such as segmentation, classification and completion. For what concerns geometric completion, most works are focused on either point cloud or volumetric data. Among them, the characteristic of having an explicitly defined local neighbourhood makes volumetric data easier to process with 3D convolutions \cite{7, 41, 42}. One drawback introduced by the predefined local neighborhood is the inaccuracy due to the constant resolution of the voxels, meaning that one voxel can represent several small structures.

On the other hand, point clouds have the advantage of not limiting the local resolution, although they come with their own sets of drawbacks. Mainly, there are two problems in processing point clouds: the undefined local neighborhood and unorganized feature map. Aiming at solving these issues, PointNet++ \cite{23}, PMP-Net \cite{35}, PointConv \cite{37} and PointCNN \cite{13} employ $k$-nearest neighbor search to define a local neighborhood, while PointNet \cite{22} and SoftPoolNet \cite{33} adopt the pooling operation to achieve permutation invariant features. Notably, point cloud segmentation and classification were further improved by involving $k$-nearest neighbor search to form local features in PointNet++ \cite{23} compared to global features in PointNet \cite{22}. Several variations of PointNet \cite{22} also succeeded in improving point cloud completion as demonstrated in FoldingNet \cite{43}, PCN \cite{45}, MSN \cite{16}. Other methods such as SoftPoolNet \cite{33} and GRNet \cite{39} explicitly present local neighbourhood in sorted feature map and voxel space, respectively.

This paper investigates grouping local features to improve the point cloud completion of objects and scenes. We apply these operation in encoder-decoder architectures...
which iteratively uses a feature extraction operation with the help of a set of displacement vectors as part of our parametric model. In addition, we also introduce a new pooling mechanism called neighbor-pooling, aimed at down-sampling the data in the encoder while, at the same time, preserving individual feature descriptors. Finally, we propose a new loss function that gradually reconstructs the target from the observable to the occluded regions. The proposed approach is evaluated on both object completion dataset with ShapeNet [3], and semantic scene completion on NYU [25] and CompleteScanNet [36], attaining significant improvements producing high resolutions reconstruction with fine-grained details.

2. Related works

This section focuses on the three most related fields – point cloud completion, point cloud features and semantic scene completion.

Point cloud completion. Given the partial scan of an object similar to Fig. 1, 3D completion aims at estimating the missing shape. In most cases, the missing region is due to self-occlusion since the partial scan is captured from a single view of the object. Particularly for point cloud, FoldingNet [43] and AtlasNet [11] are among the first works to propose an object completion based on PointNet [22] features by deforming one or more 2D grids into the desired shape. Then, PCN [45] extended their work by deforming a collection of much smaller 2D grids in order to reconstruct finer structures.

Through encoder-decoder architectures, ASFM-Net [38] and VRCNet [20] match the encoded latent feature with a completion shape prior, which produce good coarse completion results. To preserve the observed geometry from the partial scan for the fine reconstruction, MSN [16] and VRCNet [20] bypass the observed geometries by using either the minimum density sampling (MDS) or the farthest point sampling (FPS) from the observed surface and building skip connections. By embedding a volumetric sub-architecture, GRNet [39] preserves the discretized input geometries with the volumetric U-connection without sampling in the point cloud space. In more recent works, PMP-Net [35] gradually reconstructs the entire object from the observed to the nearest occluded regions. Also focusing on only predicting the occluded geometries, PoinTr [44] is among the first few transformer methods targeted on point cloud completion by translating the partial scan proxies into a set of occluded proxies to further refine the reconstruction.

Point cloud features. Notably, a large amount of work in object completion [11, 16, 33, 35, 39, 43, 45] rely on PointNet features [22]. The main advantage of [22] is its capacity to be permutation invariant through max-pooling. This is a crucial characteristic for the input point cloud because its data is unstructured.

However, the max-pooling operation disassembles the point-wise features and ignores the local neighborhood in 3D space. This motivated SoftPoolNet [33] to solve this problem by sorting the feature vectors based on the activation instead of taking the maximum values for each element. In effect, they were able to concatenate the features to form a 2D matrix so that a traditional 2D convolution from CNN can be applied.

Apart from building feature representation through pooling operations, PointNet++ [23] samples the local subset of points with the farthest point sampling (FPS) then feeds it into PointNet [22]. Based on this feature, SA-Net [34] then groups the features in different resolutions with KNN for further processing, while PMP-Net [35] uses PointNet++ features to identify the direction to which the object should be reconstructed. PoinTr [44] also solves the permutational invariant problem without pooling by adding the positional coding of the input points into a transformer.

Semantic scene completion. All the point cloud completion are designed to reconstruct a single object. Extending these methods from objects to scenes is difficult because of the difference in size and content. When we tried to train these methods for objects, we noticed that the level of noise is significantly increased such that most objects in the scene are unrecognizable. Evidently, for semantic scene completion, the objective is not only to build the full reconstruction of the scene but also to semantically label each component.

On the other hand, there have been a number of methods for semantic scene completion based on voxel grids that was initiated by SSCNet [27]. Using a similar volumetric data with 3D convolutions [7, 41, 42], VVNet [12] convolves on the 3D volumes which are back-projected from the depth images, revealing the camera view instead of a TSDF volume. Later works such as 3D-RecGAN [42] and ForkNet [32] use discriminators to optimize the convolutional encoder and decoder during training. Since 3D convolutions are heavy in terms of memory consumption especially when the input is presented in high resolution, SketchSSC [4] learns the 3D boundary of all objects in the scene to quickly estimate the resolution of the invariant features.

Although there are quite many methods targeting on volumetric semantic scene completion, there are still no related works proposed explicitly for point cloud semantic scene completion which we achieved in this paper.

3. Operators

Whether reconstructing objects or scenes from a single depth image, the objective is to process the given point
cloud of the partial scan $\mathcal{P}_{\text{in}}$ to reconstruct the complete structure $\mathcal{P}_{\text{out}}$. Most deep learning solutions [16, 20, 33, 43, 45] solve this problem by building an encoder-decoder architecture. The encoder takes the input point cloud to iteratively down-sample it into its latent feature. Then, the decoder iteratively up-sample the latent feature to reconstruct the object or scene. In this section, we illustrate our novel down-sampling and up-sampling operations that cater to point cloud completion. Thereafter, in the following sections, we use our operators as building blocks to assemble two different encoder-decoder architectures that perform object completion and semantic scene completion. We also discuss the associated loss functions.

### 3.1. Down-sampling operation

To formalize the down-sampling operation, we denote the input as the set of feature vectors $\mathcal{F}_{\text{in}} = \{ \mathbf{f}_i \}_{i=1}^{|\mathcal{F}_{\text{in}}|}$, where $\mathbf{f}_i$ is a feature vector and $|\cdot|$ is the number of elements in the set. Note that, in the first layer of the encoder, $\mathcal{F}_{\text{in}}$ is then set to the coordinates of the input point cloud. We introduce a novel down-sampling operation inspired from the Iterative Closest Point (ICP) algorithm [1, 5]. Taking an arbitrary anchor $\mathbf{f}$ from $\mathcal{F}_{\text{in}}$, we start by defining a vector $\mathbf{\delta} \in \mathbb{R}^{D_n}$. From the trainable variable $\mathbf{\delta}$, we find the feature closest to $\mathbf{f} + \mathbf{\delta}$ and compute the distance. This is formally formulated as a function

$$d(\mathbf{f}, \mathbf{\delta}) = \min_{\forall \tilde{\mathbf{f}} \in \mathcal{F}_{\text{in}}} \| (\mathbf{f} + \mathbf{\delta}) - \tilde{\mathbf{f}} \|$$

where $\mathbf{\delta}$ represents a displacement vector from $\mathbf{f}$. Multiple displacement vectors are used to describe the local geometry, each with a weight $\sigma \in \mathbb{R}$. We then assign the set as $\{ (\mathbf{\delta}_i, \sigma_i) \}_{i=1}^s$ and aggregate them with the weighted function

$$g(\mathbf{f}) = \sum_{i=0}^{s} \sigma_i \tanh \frac{\alpha}{d(\mathbf{f}, \mathbf{\delta}_i) + \beta}$$

where the constants $\alpha$ and $\beta$ are added for numerical stability. Here, the hyperbolic tangent in $g(\mathbf{f})$ produces values closer to 1 when the distance $d(\cdot)$ is small and closer to 0 when the distance is large. In practice, we can speed-up (1) with the $k$-nearest neighbor search for each anchor. A simple example of this operation is depicted in Fig. 2. This illustrates the operation in the first layer where we process the point cloud so that we can geometrically plot a feature in $\mathcal{F}_{\text{in}}$ with respect to $\{ (\mathbf{\delta}_i, \sigma_i) \}_{i=1}^s$.

Furthermore, to enforce the influence of the anchor in this operation, we also introduce the function

$$h(\mathbf{f}) = \rho \cdot \mathbf{f}$$

that projects $\mathbf{f}$ on $\rho \in \mathbb{R}^{D_n}$, which is a trainable parameter. Note that both functions $g(\cdot)$ and $h(\cdot)$ produce a scalar value.

Thus, if we aim at building a set of output feature vectors, each with a dimension of $D_{\text{out}}$, we construct the set as

$$\mathcal{F}_{\text{out}} = \left\{ \left[ g_b(\mathbf{f}_a) + h(\mathbf{f}_a) \right] \bigg|_{b=1}^{D_{\text{out}}} \right\}_{a=1}^{|\mathcal{F}_{\text{in}}|}$$

where different sets of trainable parameters $\{ (\mathbf{\delta}_i, \sigma_i) \}_{i=1}^s$ are assigned to each element, while different $\rho$ for each output vector. Moreover, the variables $s$ in (2) and $D_{\text{out}}$ in (4) are the hyper-parameters. We label this operation as the feature extraction.

**Neighbor pooling.** The final step in our down-sampling operation is to reduce the size of $\mathcal{F}_{\text{out}}$ with pooling. However, unlike Graph Max-Pooling (GMP) [15], that takes the element-wise maximum value of the feature across all the vectors, we select the subset of feature vectors with the highest activations. Therefore, while GMP disassembles their features as part of their pooling operation, we preserve the feature descriptors from $\mathcal{F}_{\text{out}}$. From the definition of $\mathcal{F}_{\text{out}}$ in (4), we base our activation for each vector $\mathbf{f}_a$

$$A_a = \sum_{b=1}^{D_{\text{out}}} \tanh |g_b(\mathbf{f}_a)|$$

on the results of $g(\cdot)$ from (2). Thereafter, we only take the $\frac{1}{t}$ of the number of feature vectors with the highest activations.

### 3.2. Up-sampling operation

The up-sampling and pooling operations in the encoder reduce the point cloud to a latent vector. In this case, if we directly use the operation in (4), the first layer in the decoder ends up with one vector since $|\mathcal{F}_{\text{in}}|$ is one. Subsequently, all
the other layers in the decoder result in a single vector. To solve this issue, our up-sampling iteratively runs (4) so that, denoting $\mathcal{F}_{in}$ as the input to the layer, we build the set of output feature vectors as

$$\mathcal{F}_{up} = \{ F_{\text{out}} \}_u=1^{N_{up}}$$

$$= \{ [g_b^{u}(f_a) + h_b^{u}(f_a)]_{b=1}^{D_{\text{out}}} \}_{a=1,u=1}^{u=|\mathcal{F}_{in}|,u=N_{up}} \quad (6)$$

which increases the number of vectors by $N_{up}$. As a result, $\mathcal{F}_{up}$ is a set of $N_{up} \cdot |\mathcal{F}_{in}|$ feature vectors. In addition to the list of hyper-parameters in Sec. 3.1, our up-sampling operation also takes $N_{up}$ as a hyper-parameter.

4. Encoder-decoder architectures

In order to uncover the strengths of our operators in Sec. 3 (i.e., feature extraction, neighbor pooling and up-sampling), we used them as building blocks to construct two different architectures. The first directly implements our operators to build an encoder-decoder while the second takes advantage of our operators to improve the transformers derived from PoinTr [44]. We refer the readers to the Supplementary Materials for the detailed parameters of the architectures.

4.1. Direct application

The objective of the first architecture is to establish that building it solely from the proposed operators (with the additional max-pooling) can already be competitive in point cloud completion. We then propose an encoder-decoder architecture based on our operators alone as shown in Fig. 3.

Figure 3. This architecture is composed of the proposed operators to build its encoder and decoder.

The encoder is composed of four alternating layers of feature extraction and neighbor pooling. As the number of points from the input is reduced by 128 times, we use a max-pooling operator to extract a vector as our latent feature. Taking the latent feature from the encoder, the decoder is then constructed from a series of up-sampling operators, resulting in a fine completion of 16,384 points.

4.2. Transformers

The second architecture aims at showing the diversity of the operators to improve the state-of-the-art from PoinTr [44] that uses transformers. We therefore propose a transformer-based architecture that is derived from [44] and our operators as summarized in Fig. 4.

Before computing the attention mechanisms in the transformer, the partial scan are subsampled due to the memory constraint of the GPU. PoinTr [44] implements the Farthest Point Sampling (FPS) to reduce the number of points and MLP to convert the points to features. Conversely, our architecture applies the proposed operators. Similar to Sec. 4.1, this involves alternating the features extraction and neighbor pooling. Since the Fourier feature [28] and SIRENs [26] have proven that the sinusoidal activation is helpful in presenting complex signals and their derivatives in layer-by-layer structures, a positional coding based on the 3D coordinates is then added to the features. In Fig. 4, we refer this block as points-to-token. Thereafter, we use the geometry-aware transformers from [44] which produces a coarse point cloud.

From the coarse point cloud, we then replace their coarse-to-fine strategy with our operators. This includes a series of alternating feature extraction and up-sampling operators as shown in Fig. 4.
It is noteworthy to emphasize the difference between our architecture from PoinTr [44] and to understand the implication of the changes. The contributions of points-to-tokens and coarse-to-fine to the overall architecture is illustrated in Fig. 5. We can observe from this figure that the FPS from PoinTr [44] only finds the distant points while the results of our neighbor pooling sketches the contours of the input point cloud to capture the meaningful structures of the object. Notably, by looking at our sketch, we can already identify the that the object is a table. This is contrary to the random points from PoinTr [44]. Moreover, our coarse-to-fine strategy uniformly reconstructs the planar region on the table as well as its base. Later, in Sec. 7, we numerically evaluate these advantages in order to show that the individual components have their own merits.

Since we previously discussed in Sec. 3.1 the difference of our down-sampling operation against 3D-GMP [15], we become curious to see the reconstruction in Fig. 5 if we replace the FPS in PoinTr [44] with the cosine similarity and GMP of [15]. Similar to PoinTr, the new combination selects distant points as its tokens while the table in their final reconstruction increased in size. In contrast, our tokens are more meaningful and the final results are more accurate.

5. Loss functions

Given the input point cloud \( P_{in} \) (e.g., from a depth image), the objective of completion is to build the set of points \( P_{out} \) that fills up the missing regions in our input data. Since we train our architecture in a supervised manner, we denote \( P_{gt} \) as the ground truth.

Completion. To evaluate the predicted point cloud, we impose the Earth-moving distance [9]. Comparing the output points to the ground truth and vice-versa, we end up

\[
L_{out \rightarrow gt} = \sum_{p \in P_{out}} \| p - \phi_{gt}(p) \|_2 \tag{7}
\]

\[
L_{gt \rightarrow out} = \sum_{p \in P_{gt}} \| p - \phi_{out}(p) \|_2 \tag{8}
\]

where \( \phi_{i}(p) \) is a bijective function that finds the closest point in the point cloud \( P_{i} \) to \( p \).

Order of points in \( P_{out} \). After training with (7) and (8), we noticed that the points in the output reconstruction are ordered from left to right as shown in Fig. 6(b). We want to take advantage of this organization and investigate this behavior further. Assuming the idea that, among the points in \( P_{out} \), we are confident that the input point cloud must be part of it, we introduce a loss function that enforces that the first subset in \( P_{out} \) is similar to \( P_{in} \). We formally write this loss function as

\[
L_{order} = \sum_{p \in P_{in}} S(\theta_{out}(p)) \cdot \| p - \phi_{out}(p) \|_2 \tag{9}
\]

where \( \theta_{out}(p) \) is the index of the closest point in \( P_{out} \) based on \( \phi_{out}(p) \) while

\[
S(\theta) = \begin{cases} 
1, & \text{if } \theta \leq |P_{in}| \\
0, & \text{otherwise}
\end{cases} \tag{10}
\]

is a step function that returns one if the index is within the first \( |P_{in}| \) points.

When we plot the results with \( L_{order} \) in Fig. 6(c), we noticed that the order in \( P_{out} \) moves from the observed to the occluded. In addition, fine-grained geometrical details such as the armrest of the chair are visible when training with \( L_{order} \); thus, improving the overall reconstruction.

Semantic scene completion. In addition to the architecture in Sec. 4 and the loss functions in (7), (8) and (9) for completion, a semantic label is added to each point in the predicted cloud \( P_{out} \). Given \( N_{c} \) categories, we denote the
label for each point as a one-hot code \( l_i = [l_{i,c}]_{c=1}^{n_c} \) for the \( i \)-th point in \( P_{out} \) and the \( c \)-th category. Since training is supervised, the ground truth point clouds are also labeled with the semantic category.

After establishing the correspondence between the predicted point cloud to the ground truth in (7) in training, we also extract the ground truth semantic label \( \hat{l}_i \). It then follows that the binary cross-entropy of the \( i \)-th point is computed

\[
\epsilon_i = -\frac{1}{N_c} \sum_{c=1}^{N_c} \hat{l}_{i,c} \log l_{i,c} + (1 - \hat{l}_{i,c})(1 - \log l_{i,c})
\]  

and formulate the semantic loss function as

\[
L_{semantic} = \frac{\gamma}{|P_{in}|} \sum_{i=1}^{|P_{in}|} \epsilon_i
\]  

where the weight

\[
\gamma = \frac{0.01}{L_{out\rightarrow gt} + L_{gt\rightarrow out}}
\]

triggers to increase the influence of the \( L_{semantic} \) in training as the completion starts to converge. Note that \( \gamma \) is an important factor, since the output point cloud is erratic in the initial iterations, which means that it can abruptly change from one iteration to the next before the completion starts converging.

6. Experiments

To highlight the strengths of the proposed method, this section focuses on two experiments – object completion and semantic scene completion.

6.1. Object completion

We evaluate the geometric completion of a single object on the ShapeNet [3] database where they have the point clouds of the partial scans as input and their corresponding ground truth completed shape. The input scans are composed of 2,048 points while the database provides a low resolution output of 2,048 points and a high resolution of 16,384 points. We follow the standard evaluation on 8 categories where all objects are roughly normalized into the same scale with point coordinates ranging between \(-1\) to \(1\).

**Numerical results.** We conduct our experiments based on three evaluation strategies from Completion3D [29], PCN [45] and MVP [20]. Evaluating on 8 objects (plane, cabinet, car, chair, lamp, sofa, table, vessel), they measure the predicted reconstruction through the L2-Chamfer distance, L1-Chamfer distance and the F-Score@1%, respectively. Note that, in this paper, we also follow the standard protocol where the value presented for the Chamfer distance is multiplied by \(10^3\). Although Table 1 only shows the average results across all categories, we refer the readers to the supplementary materials for the more detailed comparison.

One of the key observations in this table is the capacity of our direct architecture to surpass most of the other methods’ results. Among 11 approaches, our Chamfer distance is only worse than 3 methods while our F-Score@1% is better than all of them. This therefore establishes the strength of our operators since our first architecture is solely composed of it. Moreover, our second architecture, which combines our operators with the transformer, reduces the error by 3-5% on the Chamfer distance and increases the accuracy by 4.5% on the F-Score@1%.

The table also examines the effects of \( L_{order} \) to our reconstruction. Training with \( L_{order} \) improves our results by 0.12-0.13 in Chamfer distance and 0.013-0.021 in F-Score@1%, validating our observations in Fig. 6.

**Qualitative results.** We compare our object completion results in Fig. 7 with the recently proposed methods: FoldingNet [43], PCN [45], MSN [16], SoftPoolNet [33], VRCNet [20] and PoinTr [44]. The red points in the figure highlight the errors in the reconstruction. All the approaches reconstructs a point cloud with 16,384 points with the excep-
In addition to the qualitative results, we also examine the failure cases in Fig. 8. Most of them are objects with unusual structures like the car without the wheels. Another issue is when there is an insufficient amount of input point cloud to describe the object such as the chair. Notably, compared to the state-of-the-art, our reconstructions are still better in these situations.

### 6.2. Semantic scene completion

This evaluation aims at reconstructing the scene from a single depth image through a point cloud or an SDF volume where each point or voxel is categorized with a semantic class. Originally introduced for 2.5D semantic segmentation, NYU [25] and ScanNet [6], which were later annotated for semantic completion by [27, 36], are among the most relevant benchmark datasets in this field. These datasets include pairs of depth image and the corresponding semantically labeled 3D reconstruction.

#### Semantic scene completion with voxels.

NYU are provided with real scans for indoor scenes which are acquired with a Kinect depth sensor. Following SSCNet [27], the semantic categories include 12 classes of varying shapes and sizes: empty space, ceiling, floor, wall, window, chair, bed, sofa, table, tvs, furniture and other objects.

Since the other point cloud completion do not handle semantic segmentation, we start our evaluation by comparing with the voxel-based approaches which perform the both the completion and the semantic segmentation such as [2, 4, 10, 12, 14, 17, 27, 32, 46]. Considering that the volumetric data evaluates through the IoU, we need to convert our point clouds to voxel grids to make the comparison.

One of the significant advantage of point clouds over voxels is that we are not constrained to a specific resolution. Since most method evaluate on $60 \times 36 \times 60$, we converted our point cloud to this resolution. Our approach achieves competitive average IoU of 42.4% which is better than all the other methods except for SISNet [2]. However, it is noteworthy to mention that our method faces additional errors associated to the conversion from point cloud to voxels.

#### Table 2. Semantic scene completion on NYU [25] dataset. The value in resolution ($x$) is the output volumetric resolution which is $x \times 0.6x \times x$.

<table>
<thead>
<tr>
<th>Method</th>
<th>Resolution</th>
<th>Average IoU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lin et al. [14]</td>
<td>60</td>
<td>12.0</td>
</tr>
<tr>
<td>Geiger and Wang [10]</td>
<td>60</td>
<td>19.6</td>
</tr>
<tr>
<td>SSCNet [27]</td>
<td>60</td>
<td>30.5</td>
</tr>
<tr>
<td>VVNet [12]</td>
<td>60</td>
<td>32.9</td>
</tr>
<tr>
<td>SaTNet [17]</td>
<td>60</td>
<td>34.4</td>
</tr>
<tr>
<td>ForkNet [32]</td>
<td>80</td>
<td>37.1</td>
</tr>
<tr>
<td>CCPNet [46]</td>
<td>240</td>
<td>38.5</td>
</tr>
<tr>
<td>SketchSSC [4]</td>
<td>60</td>
<td>41.1</td>
</tr>
<tr>
<td>SISNet [2]</td>
<td>60</td>
<td>52.4</td>
</tr>
</tbody>
</table>

### Table 1. Evaluation on Completion3D [29], PCN [45] and MVP [20] datasets with their corresponding metrics for the object completion task.

<table>
<thead>
<tr>
<th>Method</th>
<th>Resolution</th>
<th>Average IoU</th>
</tr>
</thead>
<tbody>
<tr>
<td>FoldingNet [43]</td>
<td>19.07</td>
<td>14.31</td>
</tr>
<tr>
<td>TopNet [29]</td>
<td>14.25</td>
<td>12.15</td>
</tr>
<tr>
<td>PCN [45]</td>
<td>18.22</td>
<td>9.64</td>
</tr>
<tr>
<td>GRNet [39]</td>
<td>10.64</td>
<td>8.83</td>
</tr>
<tr>
<td>ECG [19]</td>
<td>9.21</td>
<td>8.51</td>
</tr>
<tr>
<td>NSFA [47]</td>
<td>9.13</td>
<td>8.29</td>
</tr>
<tr>
<td>SCRNN [31]</td>
<td>8.12</td>
<td>8.38</td>
</tr>
<tr>
<td>PoinTr [44]</td>
<td>9.22</td>
<td>8.38</td>
</tr>
<tr>
<td>ASFNet [38]</td>
<td>6.68</td>
<td>7.96</td>
</tr>
<tr>
<td>Ours (Direct)</td>
<td>8.35</td>
<td>8.46</td>
</tr>
<tr>
<td>--without $L_{order}$</td>
<td>8.47</td>
<td>8.59</td>
</tr>
<tr>
<td>--input $P_o$</td>
<td>5.11</td>
<td>5.37</td>
</tr>
<tr>
<td>Ours (Transformer)</td>
<td>6.64</td>
<td>7.96</td>
</tr>
<tr>
<td>--without $L_{order}$</td>
<td>6.74</td>
<td>8.09</td>
</tr>
<tr>
<td>--input $P_o$</td>
<td>4.46</td>
<td>4.95</td>
</tr>
</tbody>
</table>

Since FoldingNet and PCN take advantage of their mathematical assumption where they rely on deforming one or more planar grids, they tend to over-smooth their reconstruction where finer details such as the boat is flattened. In contrast, our method can perform better on the smooth reconstruction where finer details such as the boat is flattened. In more planar grids, they tend to over-smooth their reconstruction on the boat. However, they produce more errors which is highlighted in the unconventional lamp or chair. Overall, our reconstructions are closer to the ground truth.

Failure cases. In addition to the qualitative results, we also examine the failure cases in Fig. 8. Most of them are objects with unusual structures like the car without the wheels. Another issue is when there is an insufficient amount of input point cloud to describe the object such as the chair. Notably, compared to the state-of-the-art, our reconstructions are still better in these situations.

![Figure 8. Examples of the failure cases in object completion.](image-url)
Table 3. Evaluation on CompleteScanNet [36] and NYU [25] dataset for scene completion, measuring the average Chamfer distance trained with L2 distance (multiplied by 10^3) with the output resolution of 16,384.

<table>
<thead>
<tr>
<th>Method</th>
<th>CompleteScanNet</th>
<th>NYU</th>
</tr>
</thead>
<tbody>
<tr>
<td>FoldingNet [43]</td>
<td>11.25</td>
<td>14.66</td>
</tr>
<tr>
<td>PCN [45]</td>
<td>8.19</td>
<td>9.98</td>
</tr>
<tr>
<td>MSN [16]</td>
<td>7.28</td>
<td>8.65</td>
</tr>
<tr>
<td>SoftPoolNet [33]</td>
<td>8.27</td>
<td>9.29</td>
</tr>
<tr>
<td>GRNet [39]</td>
<td>4.56</td>
<td>5.80</td>
</tr>
<tr>
<td>VRNet [20]</td>
<td>4.29</td>
<td>5.45</td>
</tr>
<tr>
<td>PointTr [44]</td>
<td>5.08</td>
<td>5.92</td>
</tr>
<tr>
<td><strong>Ours (Direct)</strong></td>
<td><strong>3.17</strong></td>
<td><strong>4.72</strong></td>
</tr>
<tr>
<td><strong>Ours (Transformer)</strong></td>
<td><strong>3.04</strong></td>
<td><strong>4.38</strong></td>
</tr>
</tbody>
</table>

Table 4. Mix-and-match evaluation on different backbone attached to different coarse-to-fine methods for object and scene completion. The originally proposed combinations are marked in yellow.

<table>
<thead>
<tr>
<th>Backbone</th>
<th>Coarse-to-Fine</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>deform</td>
</tr>
<tr>
<td>MSN [16]</td>
<td>7.28</td>
</tr>
<tr>
<td>PointTr [44]</td>
<td>5.48</td>
</tr>
<tr>
<td>SoftPoolNet [33]</td>
<td>10.08</td>
</tr>
<tr>
<td>GRNet [39]</td>
<td>9.25</td>
</tr>
<tr>
<td>VRNet [20]</td>
<td>8.09</td>
</tr>
<tr>
<td><strong>Ours</strong></td>
<td>4.93</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Backbone</th>
<th>Coarse-to-Fine</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>deform</td>
</tr>
<tr>
<td>PointTr [44]</td>
<td>8.38</td>
</tr>
<tr>
<td>SoftPoolNet [33]</td>
<td>11.73</td>
</tr>
<tr>
<td>GRNet [39]</td>
<td>9.12</td>
</tr>
<tr>
<td>VRNet [20]</td>
<td>10.03</td>
</tr>
<tr>
<td><strong>Ours</strong></td>
<td>8.19</td>
</tr>
</tbody>
</table>

8. Conclusion

We propose three novel operators for point cloud processing. To bring out the value of these operators, we apply them on two novel architectures that are designed for object completion and semantic scene completion. The first assembles together the proposed operators in an encoder-decoder fashion, while the second incorporates them in the context of transformers. Notably, both architectures produce highly competitive results, with the latter achieving the state of the art in point cloud completion for both objects and scenes.
References


