NeeDrop: Self-supervised Shape Representation from Sparse Point Clouds using Needle Dropping

Alexandre Boulch\textsuperscript{1} Pierre-Alain Langlois\textsuperscript{2} Gilles Puy\textsuperscript{1} Renaud Marlet\textsuperscript{1,2}

\textsuperscript{1}Valeo.ai, Paris, France \textsuperscript{2}LIGM, Ecole des Ponts, Univ Gustave Eiffel, CNRS, Marne-la-Vallée, France

Abstract

There has been recently a growing interest for implicit shape representations. Contrary to explicit representations, they have no resolution limitations and they easily deal with a wide variety of surface topologies. To learn these implicit representations, current approaches rely on a certain level of shape supervision (e.g., inside/outside information or distance-to-shape knowledge), or at least require a dense point cloud (to approximate well enough the distance-to-shape). In contrast, we introduce NeeDrop, an self-supervised method for learning shape representations from possibly extremely sparse point clouds. Like in Buffon’s needle problem, we “drop” (sample) needles on the point cloud and consider that, statistically, close to the surface, the needle end points lie on opposite sides of the surface. No shape knowledge is required and the point cloud can be highly sparse, e.g., as lidar point clouds acquired by vehicles. Previous self-supervised shape representation approaches fail to produce good-quality results on this kind of data. We obtain quantitative results on par with existing supervised approaches on shape reconstruction datasets and show promising qualitative results on hard autonomous driving datasets such as KITTI.

1. Introduction

Learning-based approaches for 3D reconstruction from sparse 3D data have recently attracted lot of interest. As opposed to classical approaches \cite{3}, such as Poisson reconstruction \cite{37}, these methods enable prior knowledge to be used to enrich the representation of information-deficient inputs, e.g., low density point clouds or partial scene views.

Most of the recent learning-based methods for shape reconstruction from point clouds fall into two categories. A first category produces an explicit or parametric representation of the shape: point cloud, voxel or mesh. For instance, some may deform a geometric primitive or a template mesh \cite{30,31}, e.g., a planar patch or a sphere. The topology of both the template and the reconstruction are thus identical, which is a significant limitation. The second category of methods operates on an implicit formulation of the shapes. These methods build a continuous function over the 3D space, either based on occupancies \cite{46}, or on signed \cite{48} or unsigned \cite{13} distance functions. They are not bound by the topology of a template but require an extra-processing step for mesh extraction. Beyond shape reconstruction, several approaches also tackle the problem of shape generation by encoding shapes in a low-dimensional latent space with a constraint on the distribution of the latent variables. All existing learning-based approaches use a certain level of supervision during training, either using the information of distance to the shape or using the knowledge of which points fall inside or outside the shape.

In this work, we propose what we believe is the first self-supervised approach for shape reconstruction from sparse point cloud. It is self-supervised in the sense that it does not need to learn from actual shapes, or even from densely sampled point clouds; sparse, noisy, and even partial point clouds are enough. And it reconstructs a shape in the sense that an actual mesh can directly be produced from the occupancy field we predict, e.g., using a Marching Cubes algorithm \cite{44}, as opposed to approaches that only produce renderings or generate points on the underlying surface \cite{8}.

Like some other methods, we build a shape representation in a latent space and predict an implicit occupancy field. An actual surface can then be easily extracted as the zero-level set of the function. However, unlike most existing methods, we only use point clouds as input for learning, instead of meshes. We thus do not have to worry about shape watertightness, which is a major concern for occupancy-based methods training on meshes because most shape datasets collect good-looking but actually ill-formed meshes, thus requiring a significant preprocessing. Furthermore, we show that our method can deal with highly sparse input point clouds without surface supervision, such as point clouds captured by Lidars on moving vehicles, whereas all existing methods that use point clouds as input (except \cite{8}) assume that points are dense enough so that the distance to the shape (supervision) can be closely ap-
proximated by the distance to the point cloud, which leads to failures to learn from sparse inputs.

Our method is inspired by Buffon’s needle problem [41] where one wants to compute the probability that a needle dropped on a wooden floor with parallel strips lies across two strips. Similarly, we drop (sample) needles on the point cloud such that needles built on input points have a high probability of crossing the surface, while needles constructed a bit away from input points have a low probability. Our main contributions are:

- a new loss for self-supervised shape reconstruction formulated via needle dropping on the point cloud;
- the use of this loss to learn only from point clouds how to predict shape representations;
- a overall method that can intrinsically deal with highly sparse and partial point cloud, at train and test time;
- the validation of our method on both synthetic data and real point clouds for which no supervision is possible.

The paper is organized as follows: section 2 presents related work; section 3 describes the loss function and the network used for shape reconstruction; section 4 presents experiments showing the performance of our method and its comparison with competing techniques.

2. Related work

Representing shapes, with applications such as reconstruction and generation, has been widely studied. This section only presents learning-based methods; classical (non-learning-based) methods are surveyed in [3]. We classify related work based on the type of shape representation used and on the level of supervision used at training time.

Point clouds are a common way to represent a shape. They can be obtained, e.g., directly from depth sensors, or via photogrammetry. Pioneered by PointNet-based architectures [51, 53], learning-based methods have reached the state of the art in multiple tasks such as classification and semantic segmentation [6, 42, 61]. Point clouds have also been successful for shape generation from images [23, 33, 50] or from a prior distribution [68]. Yet, a point cloud remains a sparse representation of the underlying surface and, when generating points, their number is often a fixed parameter.

Voxels allow to directly adapt techniques developed on 2D pixels to 3D data. They are used in many tasks ranging from classification and semantic segmentation [45, 52, 66] to shape generation [60, 64], representation [54, 36, 55] or completion [18, 59]. Represented as an occupancy grid, the reconstructed surface however suffers from quantization effects, which can be mitigated using truncated signed distance functions [16, 18, 40, 57, 59].

Besides, using voxels may rapidly lead to a high memory consumption as their number grows cubically with the size of the scene. A greater accuracy, with finer voxel grids, may be obtained at the cost of a slow training process [15, 63, 66] or by integrating surface extraction for occupancy [43]. Sparse convolutions [14, 27, 28] or multi-resolution approaches [32, 64] such as octrees [56, 57, 60] can be used to reduce the memory footprint and scale to a complete scene [69].

Meshes describe a shape as a set of vertices and faces. They are the representation of choice in computer graphics and computer-aided design as they are very memory efficient and easily allow geometric operations and rendering with texture. Geometric deep learning [7] exploits their graph structure for classification and segmentation [20, 24].

A sub-category of methods operating on meshes deform one or several shape templates [31, 49, 54, 65]. While giving good results when the templates and the shapes are relatively similar, the resulting meshes necessarily have the same topology as the templates, which limits the complexity of the shapes that can be modeled. Besides, they may suffer from self-intersections caused by wide deformations. Template deformation can also be mixed with voxels, to refine a coarse reconstruction [26]. Other approaches directly predict sets of vertices and faces [17]. The result may not be continuous and can require additional mesh fixing steps.

Implicit representations model a closed shape via a continuous function of the 3D space. An actual surface is then extracted as a levelset of the function (or its gradient).

Most existing approaches associate to each point \( x \in \mathbb{R}^3 \) a signed distance \([1, 2, 11, 29, 47, 48]\) or an occupancy value \([12, 21, 25, 46]\). Such implicit representations lead to the reconstruction of closed surfaces (with an inside and an outside) without template-topology limitations. In this work, the network is trained to output an occupancy value indicating on which side of the shape a query point is.

Unsigned distance fields [8, 13] have the advantage to be able to model open surfaces too. But they do not directly yield a mesh; they generate a dense point cloud, that is then given to a meshing algorithm to generate an actual surface.

Some hybrid methods predict a set of convex polytopes with one implicit function per polytope, to produce a piece-wise representation of a shape [39, 62]. Others reconstruct a globale mesh by predicting voxel occupancy as well as a local mesh configuration [43].

Supervision level is key differentiating factor too. The full supervision of the real, signed distance to the shape (Fig. 1(a)) is typically tackled by a regression loss [43, 48]. Full supervision of the occupancy information (Fig. 1(b)) leads to networks predicting if a point is inside or outside the shape [12, 46]. A weaker supervision only provides the unsigned distance to the shape [13] (Fig. 1(c)). Finally, in the fully self-supervised setting, which is used in this work, only the distance to the input points is available, thus providing only an approximate distance to the shape (Fig. 1(d)).

Several methods fall in this last category. SAL [1] learns a sign-agnostic distance and a specific network initialization favors a sign change when crossing the surface, then allow-
formity cannot be controlled though. A separate meshing
a number of extra parameters), with a density whose uni-
the underlying surface using stochastic gradient ascent (and
shape gradient field that can be used to sample points on
it actually predicts a
easily and directly produce a mesh. It actually predicts a
although the paper talks of “surface extraction”, it does not
when trained on a collection of shapes with a very small
(No code is available for SALD.)
ments), much more like in Fig. 1(d). Yet, the sparse point
cloud setting is investigated in SAL [1] for single shape
ization and regularization, with the same caveat as SALD.
In contrast, OccNet [46] traces rays of possibly infinite
length to close shapes, while our needles are bounded.
DeepSDF [48] has a kind of needle to sign a distance \( \pm \eta \)
with respect to a viewpoint, whereas our needles have uniform
orientations and apply to self-supervised surface reconstruction.

3. Self-supervised reconstruction

3.1. Implicit shape representation

Like previous work aiming at predicting an implicit
shape representation, our goal is to approximate the ideal
function \( f_0^S \) such that some \( \alpha \in \mathbb{R} \) level-set of \( f_0^S \) is the
surface \( S \) of the target shape \( S \):

\[
S = \{ x \in \mathbb{R}^3 | f_0^S(x) = \alpha \}
\]

For signed distance estimation [1][48], \( \alpha = 0 \): negative
values are inside the shape (by convention), and positive
values are outside.

We consider here \( f_0^S \) to be an occupancy function,

\[
f_0^S : \mathbb{R}^3 \rightarrow \{0, 1\}
\]

such that \( f_0^S \) takes value 0 outside the shape and 1 inside the
shape. The surface of the shape is defined as the location of
discontinuity between values 0 and 1.

Given an input point cloud \( P_S \) sampled on the shape, the
objective is to estimate a function

\[
f_S : \mathbb{R}^3 \rightarrow [0, 1]
\]

such that \( f_S \approx f_0^S \). We consider that \( f_S \) belongs to the fam-
ily of function that can be predicted using a neural network.
Thus, \( f_S \) is continuous and we consider that the surface is
the 0.5-level set of the function.
3.2. Self-supervised learning from pairs of points

In other approaches, \( f_S^0 \) is known [12][46][48], or partially known [1][13] (known distance with great precision, unknown sign) at each point \( x \in \mathbb{R}^3 \). The neural network can thus be trained directly to approximate \( f_S^0 \) under supervision. On the contrary, in our self-supervised setting, \( f_S^3 \) is not directly accessible and \( P_3 \) is the only available information about \( S \). To reconstruct \( f_S^3 \) given only \( P_3 \), we take inspiration from Buffon’s needle problem.

Buffon’s needle problem. In [41], Buffon estimates the probability that the end points of a needle, dropped on a floor made of parallel strips of wood, lie on different strips.

Needle dropping for surface reconstruction. Inspired by this probability problem, we base our reconstruction strategy using needles (pairs of points), instead of points as in previous works. Let \( x \) and \( y \) be the end points of a needle “dropped” in 3D space. We will construct our self-supervised training loss based on the facts that if \( x \) and \( y \) lie on the same side of \( S \), then \( f_S(x) \) and \( f_S(y) \) shall be equal, while if \( x \) and \( y \) lie on opposite sides of \( S \), then \( f_S(x) \) and \( f_S(y) \) shall be different. It is illustrated on Figure 2.

3.2.1 Needle-based formulation

Our goal is to estimate the best, but unknown, occupancy function \( f_S^0 \). As this function takes values in \{0, 1\}, we can define a Bernoulli distribution \( B_x^0 \) at every position \( x \in \mathbb{R}^3 \). Similarly, we define the Bernoulli distribution \( B_x \) with parameter \( b_x = f_S(x) \) at the same location \( x \in \mathbb{R}^3 \).

To reconstruct the surface, our goal is now to minimise the distance between \( B_x \) and \( B_x^0 \) at every location \( x \in \mathbb{R}^3 \). But this problem is ill-posed because \( B_x^0 \) is unknown in our case. To overcome this issue, we propose to construct new random variables out of \( B_x \) and \( B_x^0 \) for multiple pairs of different locations \( x \) and \( y \). Let \( X \) and \( Y \) be two independent Bernoulli variables at location \( x \) and \( y \) drawn according to \( B_x \) and \( B_y \). The probability that \( X = Y \) satisfies

\[
b_{x,y} = b_x b_y + (1 - b_x)(1 - b_y).
\]

We can thus define a new Bernoulli distribution \( B_{x,y}^0 \) of parameter \( b_{x,y} \) for needle \( (x, y) \). We similarly define the target Bernoulli distribution \( B_{x,y}^0 \) of parameter \( b_{x,y}^0 \) using \( B_x^0 \).

Then to estimate \( b_x^0 \), i.e., reconstruct the surface, we propose to minimise the binary cross-entropy \( \text{BCE} \), i.e.,

\[
L_{\text{recons}}(x, y) = \sum_{(x, y) \in Q} \text{BCE}(b_{x,y}, b_{x,y}^0),
\]

for a finite set of well-chosen needles

\[
Q = \{(x_i, y_i) | i \in \{1, \ldots, |Q|\}\}.
\]

In practice, \( B_{x,y}^0 \) is unknown but it is enough to know if the needles \( (x_i, y_i) \) traverse the surface or not to be able to compute \( L_{\text{recons}} \) in Eq. 5. Inferring if a needle crosses the surface or not, when the needle is well chosen, is much easier than guessing if a point \( x \) is inside or outside the object.

3.2.2 Needle picking

We now describe our strategy to pick needles with end points on opposite sides or on the same side of the surface.

Surface-crossing needles (opposite-side end points). As the only information we have about the shape is \( P_S \), and considering that the surface is continuous and locally planar in the neighborhood of \( p \in P_S \), which is true at arbitrarily small scales, we drop likely surface-crossing needles as line segments centered at points of \( P_S \), thus with likely opposite-side end points. The corresponding set \( Q_{\text{opp}} \) is

\[
Q_{\text{opp}} = \{(p + h, p - h) | p \in P_S, h \sim \mathcal{N}(0, \sigma_h)\},
\]

where \( h \) is randomly sampled from the multivariate normal distribution \( \mathcal{N}(0, \sigma_h) \in \mathbb{R}^3 \) with standard deviation \( \sigma_h \), possibly depending on \( p \). The high probability of crossing the surface actually depends on curvature (see supp. mat.). Empirically, exceptions are rare enough not to confuse learning, as with datasets containing some erroneous labels.

Non-surface-crossing needles (same-side end points). We also have to sample needles not crossing the surface. We may note that if a 3D point \( p \) is not too close to \( P_S \) and if \( P_S \) is dense enough (which is the case in practice for ordinary objects sampled with as few as 300 points), then it is likely that the line segment between \( p \) and its closest point \( p' \) in \( P_S \) does not cross the surface \( S \), unless possibly near to \( p' \), including due to noise in input point cloud sampling. To simply take that into account, we consider closest points to \( P_{\text{opp}} \), where \( P_{\text{opp}} \) is the set of end points in \( Q_{\text{opp}} \), i.e., points of \( P_S \) with a slight offset in two opposite directions. To create short-enough needles that are unlikely to cross the surface and add variability in needle orientation w.r.t. the surface, we actually consider the following set of needles:

\[
Q_{\text{same}} = \{(p, p') | p \in P_{\text{same}}, p' = \text{nn}(p, P_{\text{opp}} \cup P_{\text{same}})\},
\]

where \( P_{\text{same}} \) are points sampled in 3D space and \( \text{nn}(p, P) \) is the nearest neighbor of \( p \) in \( P \) (excluding \( p \) itself).
Reconstruction loss. It would be natural to define the loss on \( Q = Q_{opp} \cup Q_{same} \) in Eq. (5). However, to account for possibly different sizes of \( Q_{opp} \) and \( Q_{same} \), we apply Eq. (5) independently on both sets. The loss is thus composed of two terms: a “data” term \( L_{opp} \) on \( Q_{opp} \) that enforces the surface to be located near \( P_S \) and a “regularization” term \( L_{same} \) ensuring side label consistency inside/outside the shape. The reconstruction loss satisfies

\[
L_{recons} = L_{opp} + L_{same} \quad (9)
\]

where

\[
L_{opp} = \frac{1}{|Q_{opp}|} \sum_{(x,y) \in Q_{opp}} \text{BCE}(b_{x,y}, b_{x,y}^0), \quad (10)
\]

\[
L_{same} = \frac{1}{|Q_{same}|} \sum_{(x,y) \in Q_{same}} \text{BCE}(b_{x,y}, b_{x,y}^0). \quad (11)
\]

Our loss, as \([1][2][3][9]\), does not ensure an occupied shape interior and void remaining space; the reverse is a solution too, yet yielding the same surface. Due to non-determinism, retraining on the same data may lead to either solution. Still, emptiness can be imposed at bounding box boundary.

3.3. Learning shape representations

From a collection of shapes provided only as point clouds, we learn a neural network \( \Phi \). Given a new point cloud \( P_S \) as input, \( \Phi \) predicts an occupancy function \( f_S \):

\[
f_S(x) = S \circ \Phi(P_S, x) \quad (12)
\]

where \( S \) is the sigmoid function to ensure \( f_S(x) \in [0,1] \).

Training Loss. Substituting Eq. (12) in the expression of the log-loss in Eq. (5) yields:

\[
\text{BCE}(b_{x,y}, b_{x,y}^0) = \log(e^{\Phi(P_S, x)} + 1) + \log(e^{\Phi(P_S, y)} + 1) - b_{x,y}^0 \log(e^{\Phi(P_S, x)+\Phi(P_S, y)} + 1) - (1-b_{x,y}^0) \log(e^{\Phi(P_S, x)+\Phi(P_S, y)} + 1) \quad (13)
\]

which can be further simplified by exploiting the fact that \( b_{x,y}^0 = 1 \) when \( (x,y) \in Q_{same} \), and \( b_{x,y}^0 = 0 \) when \( (x,y) \in Q_{opp} \). In addition, the gradient expression (see supp. mat.) is simple, benefiting from simplifications similar to the usual BCE loss for singletons.

4. Experiments

We conduct experiments on different datasets to validate the loss we propose and its use for surface reconstruction. We first describe the network used in our experiments and the experimental setup. Then, we evaluate our ability to reconstruct shapes from point clouds sampled on the entire surface (section 4.1). Finally, we show that our method can be applied to real point clouds from automotive datasets, either by direct transfer of previously learned models or by training from scratch, justifying our ability to work on highly sparse point clouds (section 4.2).

Network. We use the encoder/decoder network from [46]. The encoder is a PointNet [51] with 5 residual blocks. The decoder is a fully connected network conditioned via batch normalization on the latent code [19][22].

Parameters. Our method works both with sparse and dense point clouds; yet it is in sparse regime that it makes a difference. For all experiments, unless otherwise stated, we use a point cloud size \( |P_S| = |Q_{opp}| = 300 \) and we set \( |Q_{same}| = 2048 \). The parameter \( \sigma_h \) used to draw the vectors \( h \) in the construction of \( Q_{opp} \) is critical. A large \( \sigma_h \) distributes well the end points of both sides of the surface, increasing the chances that no needle in \( Q_{same} \) actually crosses the surface. However, too large a \( \sigma_h \) may lead to needles crossing the surface twice, which we assume does not occurs in our construction of the training loss. On the contrary, too small a \( \sigma_h \) reduces the influence of \( L_{opp} \) to a small neighborhood around the points in \( P_S \), preventing a good surface coverage and also increasing the chance of having needles in \( Q_{same} \) that accidentally cross the surface. As a rule of thumb, we set \( \sigma_h(p) = d_p/3 \), where \( d_p \) is the distance between a point \( p \) to its nearest neighbor in \( P_S \).

Training procedure. We train our model in an end-to-end fashion, with Adam [38] and a learning rate of \( 5 \times 10^{-4} \). During training, the points are either randomly sampled on the shape if an input mesh is available or picked in the original point cloud, e.g., for lidar scenes from KITTI. At test time, we predict the occupancy values on a grid and use a Marching Cubes algorithm [44] for shape extraction.

Metrics and dataset pre-processing. For each experiment, and in order to compare with previous works, we use the same dataset, same pre-processing and same metrics as used in the compared papers, as mentioned in each table caption (IoU, \( \ell_1 \) and \( \ell_2 \) Chamfer distance, worst Chamfer distance of the best \( x \%) \) of points).

4.1. Synthetic point clouds

To compare with state-of-the-art methods, we evaluate our self-supervised formulation, named NeeDrop, on point clouds sampled uniformly on closed shapes. Tab.1 and Fig.3 show our results on ShapeNet [10] and DFaust [5].

Evaluation on ShapeNet. We train our model on two configurations of ShapeNet. We first present results on the car subset (Tab.1(a)) as in [12][13], where the raw meshes are closed using the pre-processing of [67].

In Tab.1(b), we extend the experiment to all ShapeNet categories of [46]; qualitative examples are presented in Fig.3. For each compared method, we highlight its level of supervision, on distance and/or sign/occupancy. Unlike NeeDrop, all other methods (except ShapeGF) have a certain level of supervision. Despite the absence of supervision
in NeeDrop, we obtain competitive results on both benchmarks. We are on par with DMC \cite{43}, and only outperformed by OccNet \cite{46}, among those methods. ShapeGF has a lower Chamfer distance but outputs noisy point clouds (Fig.3(a)) from which constructing a mesh in not straightforward. In contrast, the occupancy predicted by NeeDrop allows for direct mesh extraction. (Trying to compare with IGR \cite{29} on ShapeNet failed because we could not find parameters to reconstruct acceptable surfaces.)

**Evaluation on DFaust.** When comparing on DFaust (Tab.1(c)), we include results for two variants of DeepSDF\cite{48} reported in SAL \cite{1}. The problem is that DeepSDF learns from a signed distance to a closed shape, whereas DFaust meshes are open. In the first variant DeepSDF\textsuperscript{\textdagger}, the sign of the distance is computed based on the oriented normals provided with the scans; in the second variant DeepSDF\textsuperscript{\textdouble dagger}, the normals are estimated locally with Jets\cite{9} and oriented consistently using minimal spanning trees\cite{1}.

All methods reported in \cite{1} were trained on very dense point cloud (16k points), while we target the much more difficult case of learning from low resolution point clouds. We thus also trained SAL\cite{1} with the same amount of point as NeeDrop (300 pts), with supervision on the distance to the shape as well as without supervision, i.e., replacing the distance to the surface by the distance to the point cloud. Results in Tab.1(c) with identical training settings (batch size 32, 2048 query pts) show that, as long as distances are measured w.r.t. the real surface, the performance of SAL degrades gracefully when the number of training points goes down from 16k to 300. However, when the distance to the surface has to be approximated from the training points, SAL fails to produce meshes when training only on 300 pts (and even on 1k pts). In fact, SAL may yield *unsigned* distance functions, which are also solutions to the loss; Marching Cubes then produce no surface. The specific network initialization of SAL tries to make it less likely, but unsigned distances still seems to be preferred solutions for sparse point clouds without exact distance supervision. The likely reason is that SAL solves an ill-posed problem: both the signed and the unsigned distance to the shape are solutions to the optimization problem. Obtaining a signed function is only favored by a particular initialization of the network, but not explicitly enforced during optimisation. In the self-supervised scenario with sparse point clouds, no regularisation prevents the implicit function to change sign between two points, where the actual surface should be, as the actual distance to the surface is not available. In contrast, NeeDrop enforces the surface to be supported by $P_S$ and does not require any well-designed initialization.

SALD \cite{2} does not offer code. Yet we expect sensitivity to the regularization weight for sparse data as a higher value is required to connect distant points with an iso-surface.

For IGR \cite{29}, we ran an experiment with 300 points as input for training (default is 8k pts). After a good training start, we observed divergence around epoch 200. We also used the the pre-trained model (trained with 8k points,
Figure 3. Qualitative comparison of NeeDrop to state-of-the-art methods, in ShapeNet, DFaust, and KITTI.
tested with 16k) provided by the authors. IGR produces artifacts around the shape (see Fig.3), which explains its worse quantitative performance in terms of Chamfer distance. However, we observe that it is the model which qualitatively recovers the best the details of the shapes.

**Finetuning with a smaller \( \sigma_h \).** This parameter is critical in our method. Its default value ensures a good convergence for all datasets we experimented with. Yet, small details may be lost with a too large \( \sigma \). For all datasets we experimented with, we observed that it is the model which qualitatively recovers the best the details of the shapes.

**Fixed-point training.** For comparison purposes, we sample new points on each shape at each epoch as in previous work. Yet, we also test our method with the same fixed 300 points sampled on each shape (NeeDrop-FP in Tab.1(c)) and observe a significant improvement. Fig.3 visually illustrates the improvement on a sample from the DFaust dataset. With the initial model, hands and feet, i.e., thin surfaces, are estimated with surface blobs, which are reduced after finetuning. We also observe the appearance of the belly button and the eyebrows. Besides, our reconstruction with 300 pts is almost on par with SAL with 16,384 pts. See the supp. mat. for a study of needle crossing validity for varying \( \sigma_h \).

**KITTIPedestrians.** As a last experiment on KITTI, we tried a very challenging target for shape reconstruction: pedestrians. Unlike for cars, we cannot symmetrize the input point cloud and must learn directly on the raw point cloud. In addition, pedestrian shapes are much more diverse than cars, which are globally convex. We see on Fig.4 that, despite the task difficulty, our method is able to recover the coarse shape of a pedestrian. Our failure to capture human limbs on KITTI is not due to \( \sigma_h \), as our method works well on DFaust. It is mostly due to partial views, that cannot be symmetrized, unlike cars; it seems there are then many solutions, centered around merged limbs. Other factors are morphology variety, clothing and hand-carried items, while DFaust only features 10 different people in underwear.

### 5. Conclusion

In this study, we investigate self-supervised shape reconstruction and representation. To this end, we consider “needles” dropped in 3D space around the shape, for which we are able to estimate if they cross the surface or not. We also define a loss on these line segments suitable for neural network training. The resulting approach, NeeDrop, is the first totally self-supervised approach for learning an implicit occupancy function from a collection of shape available only as sparse point clouds. We show that our method is competitive with state-of-the-art supervised or partially supervised reconstruction methods. We conduct qualitative comparisons on datasets with ground-truth meshes and qualitative experiments on the challenging KITTI dataset. We successfully reconstruct plausible shapes from partial point cloud of cars and show promising results on the very challenging pedestrian category.
References


