SPAGHETTI: Editing Implicit Shapes Through Part Aware Generation

AMIR HERTZ, Tel Aviv University, Israel
OR PEREL, Tel Aviv University and NVIDIA, Israel
RAJA GIRQYES, Tel Aviv University, Israel
OLGA SORKINE-HORNUNG, ETH Zurich, Switzerland
DANIEL COHEN-OR, Tel Aviv University, Israel

1 INTRODUCTION

In recent years, there is a surge of interest in applying neural implicit fields to represent 3D shapes and scenes. By design, such parameterizations do not limit the shape resolution, thereby allowing to faithfully recover the underlying continuous surface or volume.

The learned nature of neural implicit fields also promotes them as naturally compressed representations, capable of capturing high resolution details with low memory cost. Altogether, these attributes make them an intriguing medium for developing novel generative techniques.

Most of the recent research efforts have been focused on refining the quality of represented signals [Sitzmann et al. 2020; Takikawa et al. 2021; Martel et al. 2021] or leveraging implicit representations for shape reconstruction [Erler et al. 2020; Genova et al. 2020; Chabra et al. 2020a]. While implicit surface representations are well established in classical shape modeling literature (e.g., [Cani et al. 2008; Schmidt and Wyvill 2011]), so far only little attention has been given to editing of neural implicit shapes [Hao et al. 2020]. In particular, conditioning the form of a neural implicit shape on specified user controls is not straightforward, further hindering the adoption in 3D creative applications.

In this paper, we introduce SPAGHETTI, a novel generative model that supports direct editing of neural implicit shapes. Our framework allows for part level of control by (i) applying transformations on local areas of the generated object; and (ii) mixing and interpolating segments of different shapes.

The editing power of our model comes from its dual-level disentanglement. First, our network learns to separate local part representations from each other. This is essential, as modifications to a single part should have little effect on the rest. Our network is trained to achieve this separation without explicit part supervision. Second, each part representation is factored into intrinsic and extrinsic components, respectively controlling its detailed surface geometry and embedding in 3D space. Doing so allows our learning process to
introduce local affine transformations on shape parts while keeping them within the data distribution (see Figure 1).

As illustrated in Figure 2, our architecture can be roughly divided into three steps. At the beginning of our pipeline, the Decomposition Network receives a latent shape embedding \( z \) and projects it onto a set of latent codes \( Z^d \). Each \( z^d \in Z^d \) corresponds to a distinct part of the 3D shape, with its latent representation of surface information and global transformation. Then, the Mixing Network, based on a transformer encoder architecture, processes \( Z^d \) and outputs contextual codes \( Z^m \). Finally, the implicit shape is generated by an Occupancy Network, in the form of a transformer decoder, where a query coordinate is weighted according to \( Z^m \) to output its occupancy value.

During inference, the user can control the 3D shape by modifying its components in their raw latent state. The user can modify the local extrinsics properties of each part, as well as add, remove or replace components taken from other shapes. After each editing step, the modified parts are re-composed into a new implicit shape. See examples in Figure 1.

To enable the editing of new shapes that were not seen during training, we introduce a shape inversion optimization, which finds the matching part codes for a given unseen shape. Our mid latent representation of disentangled part embeddings facilitates the extrapolation outside the training data and enables high quality inversions.

We demonstrate the effectiveness of our method using a graphical user interface, where SPAGHETTI is accelerated by an Octree, to allow for an interactive editing experience of 3D implicit shapes. Code, pre-trained models and demos will be made publicly available upon publication.

2 RELATED WORK

3D generative models. The introduction of deep generative models within the computer vision community [Kingma and Welling 2014; Goodfellow et al. 2014; Dinh et al. 2014; Radford et al. 2016; Oord et al. 2016] quickly spawned a rich line of works capable of producing visually appealing images. Despite their success on images, adapting these models to generate 3D shapes conditioned on images. These ideas were later expanded beyond voxels to the non-Euclidean domains of point clouds [Li et al. 2018; Yang et al. 2019] and meshes [Ranjan et al. 2018; Tan et al. 2018; Nash et al. 2020]. See Chaudhuri et al. [2019] for a contemporary introduction to the field.

3D part-level representation. Decomposing 3D shapes into parts was traditionally proposed as means of improving shape representation, usually to facilitate downstream tasks involving recognition, retrieval or manipulation [Hoffman and Richards 1984; Mitra et al. 2014; Huang et al. 2014]. With the rising popularity of data-driven approaches [Kalogerakis et al. 2010; Kim et al. 2013], neural architectures have also been augmented with part level segmentation as a means of enriching shape representations [Qi et al. 2016; Wang et al. 2019b].

An emerging trend promotes the encoding of shape parts in a joint latent space to facilitate better generalization of the representation to novel, unseen shapes [Nash and Williams 2017]. Other works attempt to achieve this goal by addressing the geometry and structural composition of parts separately [Gao et al. 2019]. Although these approaches are able to capture fine geometric details, they require part supervision. To avoid the need for labels, shape parts can be composed as deep hierarchies using binary-space partitions [Chen et al. 2020], recursive neural networks [Li et al. 2017, 2019; Paschalidou et al. 2020b], and Gaussian mixture models (GMM) [Achlioptas et al. 2018; Hertz et al. 2020]. Our work uses a part decomposition network that is close in spirit to [Hertz et al. 2020], but we opt for a simplified variant of a flat GMM, rather than a hierarchy.

Neural implicit shapes. Non-neural implicit representations have been developed and used in a variety of 3D applications, such as reconstruction, modeling and morphing [Cohen-Or et al. 1998; Carr et al. 2001; Turk and O’Brien 2005; Schmidt and Wyvill 2011]. In these classical works, 3D shapes are implicitly approximated through a pre-defined family of functions, or interpolated distance fields. With the advances of learning based techniques, coordinate-based neural networks gain attention as powerful parameterizations able to fit arbitrary signals, and in particular, implicit shapes. Neural implicit functions are used to capture the geometry of 3D shapes as occupancy indicator functions [Chen and Zhang 2019; Mescheder et al. 2019; Peng et al. 2020], level sets of distance fields [Park et al. 2019; Atzmon and Lipman 2020], or indirectly as volumetric radiance fields [Mildenhall et al. 2020; Zhang et al. 2020]. Pioneering works by Chen and Zhang [2019], Mescheder et al. [2019] and Park et al. [2019] show how multiple shapes can be decoded with a single network by encoding shapes as latent codes. They are able to achieve high-quality shape reconstruction with a continuous representation that learns priors from a 3D dataset.

Neural implicit representations have been expanded to hybrid representations based on spatial structures of latent codes. Peng et al. [2020], Jiang et al. [2020b] and Chabra et al. [2020b] promote the usage of local dense grids as a means of introducing an inductive bias of spatial repetitions. Martel et al. [2021] and Takikawa et al. [2021] use hierarchical octree representations to achieve faster rendering.
closest to ours. DualSDF learns mirrored coarse and fine representations per shape, using primitives and an implicit signed distance function, respectively. Their interaction goal is different from ours: they perform shape manipulation by minimizing the objective function over transformed primitive attributes. Since some of these attributes are unconstrained, primitives that are not directly manipulated by the user may still be affected by the optimization. Thus, the method cannot guarantee that global shape attributes are maintained during editing (Figure 6). In contrast, our method uses sparse, learned representations that allow for direct editing of local shape parts while remaining faithful to the global shape structure. Since our method is part-aware, it also naturally supports mixing of parts from other shapes, as well as unconditional generation of novel shapes.

3 METHOD

Our goal is to establish a framework for editing 3D shape parts represented as learned neural implicit fields.

First, as part-level labels are expensive to obtain, we would like to learn the actual part decomposition during training. Specifically, we require the parts we form to be compact but sufficiently descriptive, as over-fragmentation may complicate the interactive editing process, and under-fragmentation may limit the degrees of freedom it allows.

Second, the latent part representations need to be disentangled from each other. This is essential to allow individual part editing and shape mixing. At the same time, the aggregation of parts should form a globally coherent 3D shape.

Third, the mapping between the latent representation and their corresponding occupancy indicators should ideally be equivariant with respect to affine transformations. Such a design allows us to directly manipulate the latent shape representation of parts during

Table 1. Neural implicit shapes methods and applications.

| Method       | Inversion | Generation | Editing | Mixing |
|--------------|-----------|------------|---------|--------|
| DeepSDF [2019]| ✗         | ✓          | ✗       | ✗      |
| IM-NET[2019]    | ✓         | ✓          | ✗       | ✗      |
| OccNet [2019]     | ✓         | ✓          | ✗       | ✗      |
| LDIF [2020]       | ✓         | ✗          | ✓       | x      |
| COALESCE [2020]   | x         | x          | x       | ✓      |
| DualSDF [2020]     | ✓         | ✓          | ✓       | ✗      |
| SPAGHETTI         | ✓         | ✓          | ✓       | ✓      |
There are numerous advantages to this choice. Predominately, it allows us to train a decoder only network that (i) decomposes a shape embedding $z$ into distinct parts embeddings $z^d_j$ that correspond to a Gaussian mixture model (GMM); (ii) processes the (masked) set of $z^d$ into contextual embeddings $z^m$ and (iii) outputs an implicit shape, where a query coordinate $x$ is projected into a high-dimensional space using positional encoding and is then weighted according to $Z^m$ in order to determine the occupancy indicator $\hat{y}$. The part disentanglement is achieved using the self-supervision, provided by the GMM for re-labeling the ground truth labels $y$. The local transformation control is achieved by applying rigid transformations on the both the Gaussians and the query coordinates.

The decomposition of a given shape is achieved using the self-supervision, provided by the GMM for re-labeling the ground truth labels $y$. The local transformation control is achieved by applying rigid transformations on the both the Gaussians and the query coordinates.

Fig. 4. Method overview. Left: the network architecture. Right: a zoom-in into the part level control component.

The final building block, Occupancy Network $f_o$, decodes the contextual embeddings to binary occupancy values, essentially forming our neural implicit representation.

Note that we treat our end-to-end framework as an auto-decoder which jointly trains an implicit shape generative model and embedding vectors $Z = \{z_j\}_{j=1}^n$, corresponding to $n$ training examples. To simplify the notations, we omit the shape index $i$ for the rest of this section. In practice, we train over datasets of shapes from the same class; see further training details in A.

The full architecture of SPAGHETTI is detailed in Figure 4, and accompanies the in-depth discussions for the rest of this chapter.

3.1 Decomposition Network

Our first objective is to obtain a part-level decomposition of a given shape, which forms the basis of our part based editing. Previous works [Genova et al. 2019a; Chen et al. 2019; Hertz et al. 2020] have shown the ability of 3D generative neural networks to learn a consistent part-level decomposition across generated shapes, without using explicit part-level supervision. Similarly, SPAGHETTI utilizes this ability by conditioning the shape generation on the parts partitioning, and their manipulation.

We formulate the Decomposition Network $f_d$ as a decoder, trained to map input shape code $z$ into a GMM representation. Given a shape embedding $z$, we first split it into $p$ distinct vectors $f_d(z) = Z^d$, where $Z^d \in \mathbb{R}^{p \times \text{dmodel}}$ is a set of high dimensional parts embeddings $\{z^d_j\}_{j=1}^p$. The encoding of each part $z^d_j \in \text{dmodel}$ is further
projected to two sets of parameters: intrinsic surface geometry information \( s_j \in \mathbb{R}^d \) and extrinsic parameters represented by the Gaussian \( g_j \in \mathbb{R}^t \) (see Figure 4b, top):

\[
\begin{align*}
  s_j &= W_z x^d_j + b_s \\
  g_j &= W_d x^d_j + b_d
\end{align*}
\]

(1)

Intuitively, \( g_j \) marks the area of influence of each part \( j \), whose detailed structural information is captured by \( s_j \). One of the advantages of this representation is that across the entire dataset, similar intra-category parts are represented using the same Gaussians in a consistent way.

The Decomposition Network \( f_d \) is a multi-layer perceptron (MLP) where after the first fully connected layer, we split the embedding to \( p \) vectors and the rest of the layers are shared between the \( m \) embeddings. It is followed by the projection of \( x^d_j \) onto the pair \( s_j \) and \( g_j \) (Eq (1)). See Appendix A.2 for further implementation details.

The low dimensional Gaussian \( g_j \) is a stacked representation of the parameters: mixing weight \( \pi_j \in \mathbb{R}^1 \), center \( \mu_j \in \mathbb{R}^3 \) and factorized covariance matrix values \( U_j \in \mathbb{R}^{3 \times 3} \), \( \Lambda_j \in \mathbb{R}^3 \). The covariance matrix can be calculated using the eigendecomposition \( \Sigma_j = U_j^{-1} D_j U_j \), where \( D_j \) is a diagonal matrix with the vector \( \Lambda_j \) as its diagonal and \( U_j \) is a unitary matrix.

Given a batch of points \( X_{\text{vol}} \in \mathbb{R}^{B \times 3} \) randomly sampled inside the full shape that corresponds to the global embedding \( z \), the network \( f_d \) is trained by the GMM negative log-likelihood loss:

\[
\mathcal{L}_{\text{GMM}}(X_{\text{vol}}; \text{GMM}) = -\log p(X_{\text{vol}}|\text{GMM}),
\]

where:

\[
p(X_{\text{vol}}|\text{GMM}) = \prod_{x \in X_{\text{vol}}} \sum_{j=1}^p \pi_j \mathcal{N}(x|\mu_j, \Sigma_j).
\]

The GMM loss encourages the Decomposition Network \( f_d \) to dissipate the Gaussians over the entire shape volume, such that every randomly sampled point can be explained by at least one of the Gaussians in the mixture. Figure 3 shows examples for the GMM decomposition learned for various shapes.

### 3.2 Implicit shape composer

The rest of our network is trained to compose together the set of part embeddings \( Z^d \), to a high resolution implicit shape. Each part representation \( Z^d \in \mathbb{R}^d \) from the Decomposition Network is disentangled to extrinsic and intrinsic components, which are then reconstructed back together to form representations \( Z^d \in \hat{Z}^d \). For brevity, we defer the discussion about attribute disentanglement to Section 3.3, and resume directly from where \( Z^d \) are fed to the rest of the pipeline (Figure 4b).

Our composition begins by using the Mixing Network \( f_m \) to reinforce the representation of each part \( Z^d \) with contextual information from other parts in \( \hat{Z}^d \). The Mixing Network outputs part representations \( \hat{Z}^m = f_m(\hat{Z}^d) \), that are global aware. Then, we use Occupancy Network \( f_o \) to obtain the composed implicit function \( \hat{y} \). Given a query coordinate \( x \in \mathbb{R}^3 \), we calculate \( \hat{y} = f_o(x|\hat{Z}^m) \), e.g. attend coordinate \( x \) on contextual representations \( \hat{Z}^m \). This yields the part-aware coordinate embedding \( \hat{x} \), which we then proceed to decode for its occupancy value \( \hat{y} \).

Both networks are realized through the full Transformer architecture of Vaswani et al. [2017] where Mixing Network \( f_m \) is realized through the Transformer encoder, and \( f_o \) is a customized variant of the decoder. The Transformer is suitable for our task due to its powerful capabilities of learning contextual representations from sequences or unordered sets in various domains [Radford et al. 2018; Devlin et al. 2018; Lee et al. 2019; Zhao et al. 2021].

The Transformer encoder of the Mixing Network \( f_m \) does not use positional encoding, since we’re interested in embeddings of an unordered set. Global aware representations \( Z^m \) are obtained through a series of multi head attention layers:

\[
\text{Attention}(Q, K, V) = \text{softmax}\left( \frac{Q K^T}{\sqrt{d_k}} \right) V.
\]

(3)

Under the formulation of Vaswani et al. [2017], the queries, keys and values matrices of each head \( h \) in layer \( t \) are given by projecting respectively:

\[
\begin{align*}
  Q_h &= \hat{Z}^d W_{Q_h} \\
  K_h &= \hat{Z}^d W_{K_h} \\
  V_h &= \hat{Z}^d W_{V_h}
\end{align*}
\]

\[
W_{Q_h} \in \mathbb{R}^{d_{\text{model}} \times d_h}; \quad W_{K_h} \in \mathbb{R}^{d_{\text{model}} \times d_h}; \quad W_{V_h} \in \mathbb{R}^{d_{\text{model}} \times d_h}
\]

with dimensions \( d_h = d_v = d_{\text{model}}/h \). The final embedding of each shape part, \( \hat{z}^m \in d_{\text{model}} \), is obtained by concatenating the per-head outputs and projecting them together.

The Transformer decoder of Occupancy Network \( f_o \) uses the output from \( f_m \) and coordinates \( x \in \mathbb{R}^3 \). During training, coordinates \( x \) are sampled along with their shape occupancy label \( y \) around the surface and within a bounding volume \([-1,1]^3 \). We denote each batch of sampled pairs as \( X_{\text{surf}} = \{(x_i, y_i)\}_{i=1}^B \). Appendix A.1 for further elaborates about the sampling scheme and data preparation.

Before feeding coordinates \( x \) to the decoder attention blocks, we first project them onto a high dimensional space \( \text{PE}(x) \in \mathbb{R}^{d_{\text{pe}}} \) using a learned positional encoding layer. To avoid potential ambiguity, we clarify that positional encodings were previously mentioned in the context of Transformers as means of preserving order in sequences. Similar formulations have been discussed in the literature of Neural Implicit Fields [Tancik et al. 2020] as means of increasing the network sensitivity to coordinate based input and overcoming Spectral Bias [Rahman et al. 2019]. Our formulation refers to the latter definition, pertaining coordinate based networks, and is closer in definition to a single SIREN layer [Sitzmann et al. 2020]:

\[
\text{PE}(x) = \sin(a(W_{\text{pe}} x + B_{\text{pe}})),
\]

(4)

where \( W_{\text{pe}} \in \mathbb{R}^{d_{\text{pe}} \times d_3} \) and \( B_{\text{pe}} \in \mathbb{R}^{d_{\text{pe}}} \) are learned parameters and \( a \) is a fixed scalar. Using a learned variant allows us for an easier initialization which avoids careful tuning due to scale sensitivity issues common in deterministic PE parameterizations. [Hertz et al. 2021].

Network \( f_o \) proceeds to calculate the part-aware coordinate embedding \( \hat{x} \) with a sequence of \( T \) cross attention layers (Eq. (3)), enumerated as \( 0 \leq t < T \). From a single coordinate point of view, layer \( t \) in \( f_o \) outputs the embedding \( \hat{x}_{t+1} \), calculated by the cross

ACM Trans. Graph., Vol. 41, No. 4, Article 106. Publication date: July 2022.
attention of \( \hat{x}_i \) with \( Z^m \):
\[
q_d = \hat{x}_i W^q d; \quad K_d = Z^m W^K d; \quad V_d = Z^m W^v d
\]
\[
W^{Q_d} \in \mathbb{R}^{d_{out} \times d_k}; \quad W^{K_d} \in \mathbb{R}^{d_{model} \times d_k}; \quad W^{V_d} \in \mathbb{R}^{d_{model} \times d_{pe}}
\]
We define \( \hat{x}_0 = PE(x) \), and designate the final output as \( \hat{x} = \hat{x}_T \). Each attention layer consists of multi-head attention followed by a position-wise feed-forward network.

Unlike the classic transformer decoder, we omit the self-attention layers from the decoder. This is essential for a couple of reasons: (i) The occupancy indicator of a coordinate \( x \in X_{surf} \) should be agnostic to other coordinates we feed in the same set \( X_{surf} \) with \( x \), and (ii) We assume the amount of sampled points is considerably larger than the number of parts, e.g.: \( B \gg m \). While it is acceptable to allow quadratic dependency on \( m \), it is desirable to keep the network run-time complexity linear in \( B \).

The last part of \( f_b \) is an MLP which decodes \( \hat{x} \) to yield an occupancy indicator \( \hat{y} \). The occupancy loss is given by the binary cross entropy loss (BCE):
\[
L_{occ}(X_{surf}, Z^d) = \frac{1}{|X_{surf}|} \sum_{(x, y) \in X_{surf}} BCE(\hat{y}, y),
\]
Notice that the contextual-part \( Z^m \) is indifferent to the query coordinates. Therefore, in order to reconstruct a 3D shape from \( z \), we feed forward the mapping network and the transformer encoder once. Then, the decoder operates in parallel for multiple coordinates, while \( Z^m \) remains fixed.

3.3 Disentanglement of extrinsic attributes
We turn to introduce the extrinsic-geometry disentanglement component, which enables local transformation control over the generated shapes (Figure 4b). Recall, we have already retrieved the geometric properties for each part representation, \( x^d_j \), by projecting it to the stacked representation of Gaussian \( g_j \) (see Section 3.1). In addition, we obtained the detailed surface information representation \( s_j \).

In the following, we attempt to make embedding \( s_j \), invariant to affine transformations applied over the shape part. At the same time, we want the mapping of \( g_j \) to the shape part geometry to be equivariant with respect to affine transformations. In other words, transformations applied on \( g_j \) should be directly mapped to the decoded shape part \( j \).

To that end, we apply a random affine transformation \( T \) on \( g_j \) to obtain the transformed Gaussian \( \hat{g}_j \). We then up-project and inject it back to \( x^d_j \) by:
\[
\hat{x}^d_j = s_j + W_d \hat{g}_j + b_d.
\]

The modified set \( \hat{Z}^d = (\hat{x}^d_j)_{j=1}^n \) is then routed to the composition networks as described in Section 3.2. Finally, we apply the same transformation \( T \) on \( X_{surf} \) such that output implicit function \( \hat{y} \) learns to mimic the transformed shape.

On one hand, any extrinsic attributes, i.e., part location and orientation, that might be concealed in \( s_j \) are now irrelevant for the reconstruction of \( \hat{y} \). This is true since transformation \( T \) is applied only on \( g_j \). On the other hand, \( \hat{g}_j \) does not contain any intrinsic surface geometry information by construction. It is extracted from the low dimensional embedding \( g_j \), which holds the parameters of a single Gaussian. Thus, disentanglement of extrinsic and intrinsic geometric attributes is achieved.

3.4 Part-level disentanglement
Ideally, we would like part representations \( Z^d \) to contain only local part information, and contextual representations \( Z^m \) to be global-aware. The former is required to allow an intuitive part-editing mechanism, where the latter is crucial to have a high-quality reconstruction of the shape \( \hat{y} \).

Even though, each part embedding \( x^d_j \) corresponds to a single shape Gaussian, there is no guarantee that additional global information will not leak between different part embeddings of the same shape. Indeed, such “leaks” may harm the quality of local editing and our ability to mix parts between shapes (see Section 4.4).

To overcome this problem, we augment each training iteration with an additional forward pass, which promotes \( Z^d \) to contain local information and better separate the Gaussians area of effect. Different to before, we carefully select a subset of part embeddings, denoted as \( Z^{b-} \subset Z^d \). Specifically, we’re interested in choosing part representations that may contain mutual knowledge about each other, which is more common with Gaussians that are proximate or potentially overlap. We therefore randomize a direction vector \( u \in \mathbb{R}^3 \), and sort all Gaussian centers in that direction. Then, we choose \( L \) sequential Gaussians whose part representations constitute \( Z^{b-} \).

From here, we proceed to reconstruct \( \hat{y} \) as usual. We expect that the output implicit shape of this forward pass will contain only shape parts that are governed by representations \( Z^{b-} \).

Since we do not have direct supervision to guide this optimization, we utilize the clustering induced by the GMM of the sampled points \( X_{surf} \) to generate self-supervised labels. We assign each coordinate \( x \in X_{surf} \) to the Gaussian that maximize its expectation:
\[
Cluster(x|GMM) = \arg \max_j \pi_j N(x|\mu_j, \Sigma_j),
\]
then, we re-label occupancy \( y \) as:
\[
y^- = \begin{cases}
  y, & \text{if } Cluster(x|GMM) \in I(\mathbb{Z}^{b-}) \\
  0, & \text{otherwise,}
\end{cases}
\]
where \( I(\mathbb{Z}^{b-}) \) are the indices of the Gaussians in \( \mathbb{Z}^{b-} \). Intuitively, we relabel the outside-coordinates, e.g.: the coordinates that are clustered outside the Gaussians of \( \mathbb{Z}^{b-} \), as “non-occupied”. The part-level disentanglement loss is then given by an occupancy loss term calculated over the modified set:
\[
L_{dis}(X_{surf}, \mathbb{Z}^{b-}, GMM) = L_{occ}(X_{surf}^r, \mathbb{Z}^{b-}).
\]
3.5 Training loss function

The complete loss term for our network is given by the terms discussed so far:

\[
\mathcal{L}_{\text{SPAGHETTI}} = \mathcal{L}_{\text{GMM}} + \mathcal{L}_{\text{occ}} + \mathcal{L}_{\text{dis}} + \gamma \|z\|_2^2, \quad (10)
\]

where \(\gamma\) is a hyperparameter controlling the loss weight. We also apply regularization \(\|z\|_2^2\) per global shape embedding, as advocated by previous auto-decoder works [Bojanowski et al. 2018; Park et al. 2019]. The latent regularization promotes the shape codes to be normally distributed. That, in turn, makes the space of global shape codes easier to sample from.

3.6 Shape inversion

Our framework learns to represent shapes through an auto-decoder [Park et al. 2019]. To allow editing of a new shape that is not part of the training data, we first have to match it with a shape embedding \(z\). Our goal is to acquire part embeddings \(Z^d\) such that the generated implicit shape is as close as possible to the new given shape. We suggest a simple two-steps optimization process to find these embeddings.

In the first step, we begin with a randomly initialized code:

\(z \sim \mathcal{N}(\mu_T, \Sigma_T)\),

using the mean and covariance of shape embeddings seen during training. We sample points \(X_{\text{vol}}\) inside the new given shape, and use \(f_g\) with frozen weights to obtain \(Z^d\). Specifically, \(z\) is optimized using the GMM negative log-likelihood (Eq. (2)) between the generated GMM of \(z\) and sampled points, \(X_{\text{vol}}\), inside the new given shape.

In the second optimization step, we ensure the obtained \(Z^d\) reproduces the shape accurately. Therefore, we freeze the layers of \(f_m, f_o\) and use occupancy loss (Eq. (6)) to further optimize the part embeddings \(Z^d\):

\[
\arg \max_{Z^d} \mathcal{L}_{\text{occ}}(X_{\text{cube}}, Z^m),
\]

where \(Z^m\) are given by the contextual encoder \(Z^m = f_m(Z^d)\).

Upon convergence, we can use \(Z^d\) in our interactive interface as described in the following section.

3.7 User interface for interactive shape editing

We demonstrate how a pre-trained SPAGHETTI model can be applied as an interactive editing framework. In this setting, the user can manipulate newly generated shapes, or reference shapes taken from an existing dataset. Examples of some editing operations can be seen in Figures 1, 5, 6, as well as in the supplementary video. Our coarse GMM representation is used for partitioning the generated shapes. The partition provided by the GMMs enables a simple interface for quick selection of shape parts that are highlighted together with the same color-coding.

Users can select parts from different shapes, mix them together, and assemble new shapes. Under the hood, each part selection corresponds to selection of latent vectors from \(Z^d\). These selected latents can then be combined with latent vectors of selected parts from other shapes, to form a single set of latent codes. The combined latents are forwarded through our pretrained network (as shown in Figure 4a) to synthesize newly mixed shapes.

In addition, users can select shape parts and apply affine transformations to them, which results with local deformation of the selected parts. As shown in Figure 4a, the specified transformations are applied on the Gaussian parameters of the selected part. After transformations are applied, the Gaussian representation is injected back to the part latent representation, which goes to the transformer part of our network to synthesize the new manipulated shape.

Finally, we remark on the rasterization pipeline we used for rendering implicit shapes. After each editing step, we reconstruct a mesh with the marching cube algorithm [Lorensen and Cline 1987], using a grid resolution of 256\(^3\). To reduce the number of queries through the network [Takikawa et al. 2021; Hedman et al. 2021], we backed the grid implementation with our in-house implementation of an Octree. The acceleration structure enables an interactive rate of about two seconds between each editing step. Future applications may opt to avoid the mesh conversion step and render the implicit shape directly using ray-marching techniques.

4 EXPERIMENTS

SPAGHETTI can be leveraged for various applications that include shape inversion, generation, editing and mixing. These tasks are summarized in Table 1. In this section we demonstrate the advantages of SPAGHETTI for each application and compare it to the other relevant learning based methods for implicit shapes synthesis. In addition, we conduct an ablation study to evaluate the different components used in our framework.

Our experiments are conducted on the ShapeNet dataset [Chang et al. 2015]. We use the train-test split of DeepSDF [Park et al. 2019], where the number of shapes in the train set, per category, varies from \(\sim 1k\) (lamps) to \(\sim 5k\) (tables). We train a separate network for each shape category. Further details are included in Appendix A.1.

4.1 Editing comparisons

Concerning neural implicit shapes, only few methods provide editing controls. Moreover, some of the existing methods have set different editing objectives than us. In the following, we discuss the main difference between SPAGHETTI and two contemporary methods which are closest to us: COALESCE [Yin et al. 2020] and DualSDF [Hao et al. 2020]. These works enable shape mixing and shape editing, respectively.

Shape mixing: The objective of COALESCE is to put together a given set of shape parts and output a unified shape. Their pipeline consists of three main steps: (i) A part alignment network, which outputs a transformation for each input part such that the transformed parts are aligned together. (ii) A joint synthesis network that synthesizes new implicit joint parts, which connect the separate parts. (iii) “Poisson mesh stitching” is applied over the parts and the newly formed joints. Each COALESCE network is trained over segmented parts from a single shape category. Figure 5 shows mixing examples of our method compared to the stitching operation of COALESCE, using pre-trained networks on the chairs and airplanes categories. Both methods, receive as an input a set of shape parts. Evidently, synthesizing joints as implicit functions and then stitching them automatically to existing meshes is prone to result in noisy artifacts at the joints regions. SPAGHETTI combines the latent parts
Table 2. Shape inversion comparisons. CD = chamfer distance; EMD = earth mover’s distance; ACC: mesh accuracy [Seitz et al. 2006]. In all measurements, lower score is better. All measurements are multiplied by a scale of $10^3$.

| Method         | Airplanes | Chairs | Lamps |
|----------------|-----------|--------|-------|
|                | CD$_{\text{mean/med.}}$ | EMD$_{\text{mean/med.}}$ | ACC | CD$_{\text{mean/med.}}$ | EMD$_{\text{mean/med.}}$ | ACC |
| IM-NET [2019]  | 0.435 / 0.195 | 28.96 / 25.09 | 17.541 | 0.625 / 0.436 | 33.40 / 30.53 | 22.062 |
| LDIF [2020]    | 0.634 / 0.178 | 31.50 / 20.23 | 17.237 | 0.800 / 0.425 | 29.69 / 25.65 | 16.787 |
| DeepSDF [2019] | 0.095 / 0.029 | 16.00 / 13.17 | 6.229 | 0.323 / 0.113 | 24.23 / 19.72 | 14.56 |
| DualSDF [2020] | 0.806 / 0.097 | 31.93 / 22.20 | 26.78 | 0.688 / 0.369 | 34.98 / 32.46 | 29.59 |
| SPAGHETTI     | 0.050 / 0.011 | 9.27 / 7.17 | 4.237 | 0.102 / 0.032 | 13.55 / 11.26 | 6.884 |

| Method                  | JSD$_1$ | MMD↑ | COV↑ | 1-NNA↑ | JSD$_1$ | MMD↑ | COV↑ | 1-NNA↑ | JSD$_1$ | MMD↑ | COV↑ | 1-NNA↑ |
|-------------------------|---------|------|------|--------|---------|------|------|--------|---------|------|------|--------|
| DeepSDF                 | 3.89    | 3.8 / 10.2 | 32.6 / 33.5 | 70 / 71 | 1.62    | 11.1 / 13.6 | 41.2 / 44.3 | 60 / 61 | 1.35    | 17.3 / 14.8 | 42.1 / 41.1 | 59 / 59 |
| IM-NET                  | 3.77    | 4.2 / 10.5 | 30.0 / 33.2 | 65 / 64 | 2.37    | 12.4 / 15.1 | 37.7 / 36.4 | 61 / 62 | 3.35    | 16.6 / 15.7 | 37.7 / 39.2 | 61 / 61 |
| DualSDF                 | 6.78    | 4.2 / 11.1 | 25.0 / 24.1 | 70 / 77 | 4.49    | 10.4 / 15.9 | 32.6 / 27.1 | 70 / 76 | 2.19    | 12.3 / 15.2 | 36.3 / 32.7 | 68 / 72 |
| SPAGHETTI               | 2.28    | 2.4 / 8.10 | 35.0 / 41.3 | 61 / 61 | 1.02    | 6.01 / 11.4 | 50.8 / 51.2 | 58 / 59 | 1.15    | 5.9 / 11.1 | 47.8 / 48.8 | 56 / 56 |

Table 3. Shape generation comparisons. ↑ (↓): higher (lower) is better. MMD-CD scores are multiplied by $10^3$; MMD-EMD scores are multiplied by $10^2$; JSD scores are multiplied by $10^2$.

Table 4. Disentanglement ablation. Correspondence (Cor.) measures the IoU (%) between distinct parts in our coarser GMM representation to the output implicit shape. Coverage (Cov.) measures the IoU of the whole output implicit shape and the union of its distinct parts. For both, higher score is better.

Table 5. Mixing ablation. Segmentation (Seg.) measures the Jensen-Shannon divergence (multiplied by $10^3$) between segmented parts of the generated mixed shapes to the ground truth input parts. Area measures the surface area error (%) between them. For both, lower score is better.

4.2 Shape Inversion

We evaluate the shape inversion quality of our method on 3 shape categories from the Shapenet dataset [Chang et al. 2015]: airplanes, chairs and lamps, using the train-test split of DeepSDF [Park et al. 2019]. We compare our results to other generative and reconstruction methods that output implicit shapes: IM-NET [Chen and Zhang 2019], LDIF [Genova et al. 2019a], DeepSDF [Park et al. 2019] and DualSDF [Hao et al. 2020]. We train a separate network for each
Input parts  SPAGHETTI  COALESCE

Fig. 5. Mixing comparison. On the left are the input parts supplied to our method and COALESCE [Yin et al. 2020] in order to create a unified novel mixed shape. While COALESCE synthesizes only the joints between parts, our method synthesize the whole mixed shape.

Fig. 6. Sequential editing of implicit shapes using our method (top rows) and DualSDF [Hao et al. 2020] (bottom rows). The input edit (marked in red) provided by the user is shown between the columns.

shape category for each method using their official code and training settings.

The shape inversion process for auto-decoder based methods, DeepSDF and DualSDF, is obtained through an optimization process. They optimize a latent shape vector $z$, that minimizes the reconstruction loss:

$$\arg \min_z L_{rec}(G(x, z), y),$$

ACM Trans. Graph., Vol. 41, No. 4, Article 106. Publication date: July 2022.
where $G$ is the method's generative model, $x$ are the 3D coordinates and $y$ are their corresponding labels, i.e., the signed distances of the SDF representation. $L_{\text{rec}}$ is determined by the specific loss settings of each method.

IM-NET and LDIF use encoder-decoder architectures and achieve shape inversion by employing the encoder. IM-NET’s Occupancy Network is conditioned on a latent vector that is encoded from a 3D occupancy grid of the input shape. LDIF encodes 24 depth images from different views of the input shape and recovers an implicit shape.

**Evaluation metric.** Following prior works, the measured distances between the inverted implicit shape and the ground truth meshes are chamfer distance (mean and median), earth mover’s distance (mean and median) and mesh accuracy [Seitz et al. 2006]. The chamfer distance and earth mover’s distance are measured between 30,000 and 1024 sampled points, respectively, on the generated shape and the ground truth mesh. The mesh accuracy value $d$ is the minimal distance such that 90% of 30,000 sampled points on the generated shape are within an Euclidean distance $d$ of the surface of the ground truth shape.

The quantitative results are summarized in Table 2 and qualitative results are shown in Figure 8.

### 4.3 Shape generation evaluation

In addition to the editing capabilities of our method, we can use a pre-trained SPAGHETTI network for random, unconditional shape generation. Similar to [Bojanowski et al. 2018], we represent the latent distribution as a multivariate Gaussian distribution that best fits the latent space $Z$ of our training data. Then, for shape generation, we feed-forward a sampled vector $z$ through our network to get its corresponding shape.

Ideally, for fair evaluation, we would like to measure the quality of our generated shapes set $A$, with respect to the underlying shape distribution of the training data. However, since this distribution is unknown, we can only measure the quality of the generated shapes in $A$ with respect to some empirical distribution represented by an additional shapes set $B$. In our case, the set $B$ is composed of shapes from the training and test datasets.

In our evaluations, we randomly sampled 2048 points on each shape in $A$ and $B$. Then, we followed prior 3D generative works [Yang et al. 2019; Gal et al. 2021] and used the metrics introduced by [Achlioptas et al. 2018]:

- **The Jensen-Shannon Divergence (JSD)** measured between the voxel occupancy probability induced by all shapes in $A$ versus all shapes in $B$.
- **Coverage (Cov)** measured by the percentage of shapes in $B$ that are covered by a shape in $A$. For this evaluation we assign for each shape in $A$, its closest shape in $B$. Then, a shape in $B$ is considered covered if it is assigned by at least one shape in $A$.
- **Minimum matching distance (MMD)** measured by the average distance of each shape in $B$ and its closest shape in $A$.
- **1-nearest neighbor accuracy (1-NNA)** proposed by [Lopez-Paz and Oquab 2017]. This measurement penalizes each shape $s$ either in $A$ or $B$ whose closest shape lays in the same group as $s$.

We show results with both chamfer distance (CD) and earth mover’s distance (EMD) as the distance measures for the COV, MMD and 1-NNA metrics. We randomly generate 1000 shapes to compose the set $A$ for each method and shape category. The set $B$ is composed of randomly selected 500 shapes from the training set and 500 shapes from the test set, repeated per category.

The quantitative results are summarized in Table 3, where we compare our method to 3 other generative methods for implicit
Ground Truth  IM-NET  LDIF  DeepSDF  DualSDF  SPAGHETTI

Fig. 8. Shape inversion comparison. Uncurated results of the first two test shapes from the Chairs, Airplanes and Lamps categories of ShapeNet [2015].

shapes: IM-NET [Chen and Zhang 2019], DeepSDF [Park et al. 2019] and DualSDF [Hao et al. 2020]. We repeated our evaluation over 3 shape categories from the Shapenet dataset [Chang et al. 2015]: airplanes, chairs and tables. Additional qualitative results of random generated samples are included in appendix B.

4.4 Ablation studies

To validate our architecture and training settings, we compare our final model to two reduced variants of our method. The first variant, “SPAGHETTI no-enc”, omits the middle Mixing Network $f_m$. For the second variant, “SPAGHETTI no-dis”, the network architecture remains the same, but we omit the disentanglement loss $L_{dis}$ (Eq. (9)) from the training objective.

The results of the ablation study for shape inversion are included in Table 2. Notably, the two reduced SPAGHETTI variants achieve slightly worse inversion results. More importantly, part level control and the quality of editable shapes significantly degrades for these reduced variants. In the following we further discuss these phenomena.

Disentanglement ablation. One of the main objectives of our work is to enforce part-level disentanglement over our mid latent representation $Z^d$, such that shape manipulations achieved through modifications of latent vector $z^d_{idi}$, will result only in local changes to the output shape. Effectively, $z^d_{idi}$ should control the shape region that is most likely coming from the corresponding Gaussian $gi$. 
We evaluate SPAGHETTI’s ability to correspond the extrinsic GMM representation with disjoint, implicit shape parts the network outputs, by conducting two ablation tests.

For each training shape, representations \( \mathbf{Z}^d \) are obtained by feeding the shape through the Decomposition Network. We then employ the coarse segmentation annotations of PartNet [Mo et al. 2019]. We sample 1,666 coordinates within the volume of the shape, and assign each of them the segmentation label from PartNet.

We compare every part from PartNet, against the Gaussians obtained from \( \mathbf{Z}^d \). Each segmented point is attributed to the Gaussian that maximizes its expectation (Eq. (8)). Then, each Gaussian is mapped to a PartNet label, according to the segmented points attributed to it.

Our setup so far, automatically simulates a user’s selection of semantic parts by manually marking their corresponding Gaussians. We partitioned the mid-latent part representations \( \mathbf{Z}^{d_1} \), and their corresponding GMM to disjoint groups \( \mathbf{Z}^d = \bigcup_{i=1}^p \mathbf{Z}^{d_i} \), where the Gaussians that correspond to each subset \( \mathbf{Z}^{d_i} \) represent a distinct segmented part \( p_i \) among the \( p \) parts provided by PartNet.

Using this partitioned latent space, we measure two intersection over union (IoU) scores with respect to the 1,666 points already sampled inside the shape, and 1,666 more points uniformly sampled within the unit cube.

First, we measure the latent representations to part correspondence (Cor.) of each part separately. For each part \( p_i \), we mask out the representations in \( \mathbf{Z}^d \) of the Gaussians not associated with \( \mathbf{Z}^{d_i} \). Then, we feed the 2,666 sampled coordinates through the Occupancy Network \( f_o \), attended over masked part representations, processed by \( f_m \). We specify a label of \( y = 1 \) for each coordinate \( x \in p_i \), and \( y = 0 \) otherwise. The IoU is measured by comparing these labels with predictions obtained from the occupancy indicator \( f_o(x|f_m(\mathbf{Z}^{d_i})) \). Intuitively, a higher IoU score indicates that the latent vectors \( \mathbf{Z}^{d_i} \) are indeed responsible for the generation of the specific parts associated with their Gaussians.

In the second test we conduct, we measure the coverage IoU (Cov.) between the union of implicit shapes obtained from the distinct parts and the full implicit shape. Here, a higher score means that the union of the separate implicit functions faithfully represents the complete shape, i.e., that:

\[
\bigcup_{i=1}^p f_o(x|f_m(\mathbf{Z}^{d_i})) \approx f_o(x|f_m(\mathbf{Z}^d)).
\]

We train and evaluate the three network configurations over three ShapeNet categories with part segmentation annotations: tables, chairs and lamps. The results are summarized in Table 4.

Compared to the full SPAGHETTI architecture, the ‘SPAGHETTI no-dis’ variant, trained without the explicit disentanglement loss, yields poor correspondence between the latent part representations and the output shape.

‘SPAGHETTI no-enc’ variant, which omits the Mixing Network, achieves slightly better IoU scores for local part correspondences (Cor.) but performs worse in terms of global shape coverage (Cov.). We attribute these differences to interactions that occur within the Mixing Network, which augment the contextual representation of each part vector \( z^m \) with global information. Introducing global information requires additional capacity from the part embeddings, which may come at the expense of local part information. Nevertheless, in practice, we aim to maintain a balance between global coherency and local parts separation. We therefore find that the inclusion of the Mixing Network is crucial for synthesizing distinct parts to a globally coherent shape (Figure 9). In particular, the Mixing Network is an important backbone for the “mixing” edit-operations, e.g., synthesizing together parts from different shapes.

**Mixing ablation.** We conduct an additional ablation test, where we start with the same segmented part partition as before. Each latent code \( z^d \) and its projected Gaussian are mapped to some part \( p_i \) from PartNet, but for each shape, we replace one of the part codes \( z^d \)
with a code from another shape. The replaced code comes from the same part category, i.e.: we may replace the latent code representing a leg of a chair with code representing a leg of another chair.

A qualitative comparison for this experiment is shown in Figure 9. We observe that the settings that remove the Mixing Network (SPAGHETTI no-enc) or train without \( L_{\text{dis}} \) (SPAGHETTI no-dis) are prone to noisy artifacts. At the same time, our full settings preserve the distinct parts better, while generating well-figured mixed shapes.

Since we do not have a ground truth mixed shape to compare to, we are compelled to conduct an indirect quantitative comparison instead. Our ablation aims to give an upper bound for the quality of mixed shapes, by comparing specific attributes of the “mixed” shapes with respect to their input parts.

The first approximation bound we use, measures the difference between the surface area of the mixed shape to the surface area of the input shape. We report the percentage of that difference with respect to the surface area of the input parts. We refer to this measurement as area evaluation. Since this evaluation only gives a rough indication for the quality of mixed shapes, we suggest an additional criterion.

The second bound we employ, measures the segmentation quality of the generated shape with respect to the input parts. For this evaluation, we trained a segmentation network [Qi et al. 2017] for each shape category, and used it to estimate the surface area of each segmentation class, per shape. Then, we divide those areas by the total area of the entire shape, to obtain a distribution over the segmentation-classes. Our goal is to demonstrate this manifold is smooth in the geometric sense. Continuous interpolations between them. Our goal is to demonstrate that this manifold is smooth in the geometric sense.

Qualitative segmentation examples are shown in Figure 9 and quantitative results are summarized in Table 5 (Seg. and area evaluation).

4.5 Part level interpolation

We now demonstrate the properties of the obtained manifold that contains latent part embeddings \( Z^d \), and evaluate the quality of continuous interpolations between them. Our goal is to demonstrate that this manifold is smooth in the geometric sense.

Our evaluation does not concern global interpolation, where one can simply interpolate between different shape embeddings \( z \) (although our method can be used also for this purpose as well), but instead we focus on interpolations performed between sets of shape parts. Such interpolations are non-trivial, as given two part embeddings sets \( Z^d_1 \) and \( Z^d_2 \), we do not posses correspondences between matching part. Moreover, \( Z^d_1 \) and \( Z^d_2 \) might consist of different number of part embeddings. Therefore, we suggest an interpolation scheme throughout the attention weights computed by the cross attention layers of \( f_o \).

First, we compute all contextual part embeddings for two given shapes: \( Z^m_1 = f_m(Z^d_1) \) and \( Z^m_2 = f_m(Z^d_2) \). Then, we replace each multi head attention value with an interpolated attention weight.

Recall, the Occupancy Network uses the multi head attention formulation: \( \text{Attention}(q_d, K_d, V_d) \).

This is the Transformer decoder depicted in Eq. (5), where \( q_d \) is the query vector of \( x_t \), and \( K_d, V_d \) are the key and value matrices for \( Z^m \). For brevity, we omit the decoder notation \( d \) in the following description.

Let \( K_1, K_2 \) and \( V_1, V_2 \) represent the keys and values of \( Z^m_1 \) and \( Z^m_2 \), respectively (Eq. (5)). For a linear interpolation weight \( \alpha \in [0, 1] \), the interpolated attention is given by:

\[
\text{I-Attention}(\alpha, q, K_1, V_1, K_2, V_2) = (1 - \alpha)\text{Attention}(q, K_1, V_1) + \alpha\text{Attention}(q, K_2, V_2).
\]

From here, we continue to output the occupancy indication by attending coordinates \( x \) over the interpolated attention value.

Figure 7 illustrates some qualitative results of part level interpolation. During rendering, we highlight the interpolated parts by examining the attention weights of coordinates on the iso-surface, or vertices of a reconstructed mesh. We assign each surface coordinate \( x \) to the part that receives the most attention from \( x \):

\[
\arg \max_j \sum_{i=0}^{T} \sum_{k=0}^{K} \text{softmax} \left( \frac{q_i K^T_{ij}}{\sqrt{d_k}} \right).
\]

where aggregation is done over the transformer layers and heads. Finally, we color the vertices that are assigned to the symmetric difference indices \( I \left( Z^m_1 \triangle Z^m_2 \right) \), that is, coordinates that attend to interpolated codes.

5 CONCLUSIONS, LIMITATIONS AND FUTURE WORK

We presented a generative neural model for implicit 3D shapes, featuring local editing capabilities. Our model is part-aware, requires no part supervision, and leverages the Transformer architecture to form globally coherent shapes. The network was designed to allow editing at an interactive rate, where, as demonstrated, the user can interact with the generated model using a simple interface.

Our method relies on dual-level disentanglements. First, our model creates disentangled latent codes for disjoint generated shape parts. This allows supporting selection and mixing between different shapes.
shape parts. Second, each part embedding is a factorized representa-
ion of intrinsic and extrinsic information, which are used to conduct
local deformations over the shape.

In our work, we assume to have 3D training data that consists of
shapes with similar structure, such as chairs and airplanes. Such
structured data enables the unsupervised learning of consistent
partitions of compact parts. However, in some shape categories, this
assumption holds weakly. For example, the lamps dataset consists of
many unique lamps (Figure 3). Moreover, since we do not employ
part-level supervision, our model is agnostic to the semantics of part
shapes. Our GMM partition may over-cluster together parts into
the same Gaussian which may prevent the desired level of control.
For example, see Figure 10, where the leg of the chair is represented by
only two Gaussians, even though it consists of many sub parts.
Similarly, the model may under-cluster large parts and represent
them with multiple latent codes, for example, the back of a chair can
be represented by more than a single Gaussian. We conjecture that
these limitations can be addressed by introducing semi-supervised
annotations, or through hierarchical part decomposition [Eckart
et al. 2018; Paschalidou et al. 2020a; Hertz et al. 2020] which provides
partitioning of the input shape at various scales.

Another direction left for future work is the design of an en-
corder which predicts the latent codes for a pre-trained SPAGHETTI
model, conditioned on different input modalities. For example, when
trained on complete 3D shapes, such an encoder may accelerate the
inversion process by replacing the optimization steps with a single
feed-forward pass. In addition, by training an encoder on 3D scans or
2D images, we may leverage SPAGHETTI’s expressiveness for
means of 3D reconstruction.

In this work, we explored how learning-based techniques can
further assist future workflows of 3D modeling. In a supervised or
semi-supervised settings, where the part decomposition is guided
by instance-level labels, the performance can be further improved,
which may also lead to co-segmentation as a byproduct. In the
future, we would like to add more intuitive 3D editing tools, and
other types of interactive interfaces. For example, guiding the 3D
modeling process by 2D sketches or through textual descriptions.

ACKNOWLEDGMENTS

This work is supported in part by the European Research COUN-
cil (ERC) Consolidator grant no. 101003104 and Starting grant no.
757497, by the Israel Science Foundation grant no. 2492/20 and by
the Innovation Programme grant no. 101003104.

REFERENCES

Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. 2018. Learn-
ing representations and generative models for 3d point clouds. In International
conference on machine learning. PMLR, 40–49.
Matan Atzmon and Yaron Lipman. 2020. SAL: Sign Agnostic Learning of Shapes
From Raw Data. In IEEE/ACM Conference on Computer Vision and Pattern Recognition
(CVPR).
Gavin Barill, Neil Dickson, Ryan Schmidt, David I.W. Levin, and Alec Jacobson. 2018.
Fast Winding Numbers for Soups and Clouds. ACM Transactions on Graphics (2018).
Piotr Bojanowski, Armand Joulin, David Lopez-Paz, and Arthur Szlam. 2018. Optimizing
the Latent Space of Generative Networks. In International Conference on Machine
Learning. PMLR, 600–609.
Marie-Paule Cani, Takero Igashita, and Geoff Wyvill. 2008. Interactive Shape De-
sign. Morgan & Claypool Publishers. 78 pages. https://doi.org/10.2200/S00122ED1V01Y200806CR006
J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and
T. R. Evans. 2001. Reconstruction and Representation of 3D Objects with Radial
Basis Functions (SIGGRAPH ’01). Association for Computing Machinery, New York,
NY, USA, 67–76. https://doi.org/10.1145/383259.383266
Rohan Chabra, Jan E Lensen, Eddy Igl, Tanner Schmidt, Julian Sturag, Steven Lovegrove,
and Richard Newcombe. 2020b. Deep Local Shapes: Learning Local SDF Priors
for Detailed 3D Reconstruction. In Computer Vision – ECCV 2020. Andrea Vedaldi,
Horst Bischof, Thomas Brox, and Jan-Michael Frahm (Eds.). Springer International
Publishing, Cham, 688–695.
Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang,
Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. 2015. ShapeNet:
An information-rich 3d model repository. arXiv preprint arXiv:1512.03012 (2015).
Siddhartha Chaudhuri, Daniel Ritchie, Kai Xu, and Hao Zhang. 2019. Learning Genera-
tive Models of 3D Structures. In Eurographics.
Zhiqin Chen, Andrea Tagliasacchi, and Hao Zhang. 2020. BSP-Net: Generating Compact
Mesbes via Binary Space Partitioning. Proceedings of IEEE Conference on Computer
Vision and Pattern Recognition (CVPR) (2020).
Zhiqin Chen, Kangxue Yin, Matthew Fisher, Siddhartha Chaudhuri, and Hao Zhang.
2019. BAE-NET: Branched Autoencoder for Shape Co-Segmentation. In The
Proceedings of the IEEE/ACM International Conference on Computer Vision (CVPR).
Zhiqin Chen and Hao Zhang. 2019. Learning implicit fields for generative shape modeli-
ing. In Proceedings of the IEEE/ACM Conference on Computer Vision and Pattern
Recognition. 5993–5998.
Daniel Cohen-Or, Amir Solomonovic, and David Levin. 1998. Three-Dimensional
Distance Field Metamorphosis. ACM Trans. Graph. 17, 2 (apr 1998), 116–141.
https://doi.org/10.1145/274363.274366
Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert:
Pre-training of deep bidirectional transformers for language understanding. arXiv
preprint arXiv:1810.04805 (2018).
Laurent Dinh, David Krueger, and Yoshua Bengio. 2014. NICE: Non-linear independent
components estimation. arXiv preprint arXiv:1410.8316 (2014).
Benjamin Eckart, Kihwan Kim, and Jan Kautz. 2018. Hmr: Hierarchical gaussian
mixtures for adaptive 3d registration. In Proceedings of the European Conference on
Computer Vision (ECCV), 705–721.
Philipp Erler, Paul Guerrero, Stefan Ohrballinger, Niloy J. Mitra, and Michael
Wimmer. 2020. Points2Surf: learning implicit surfaces from point clouds. In European
Conference on Computer Vision. Springer, 108–124.
Noa Fish*, Melinos Averkiou*, Oliver van Kaick, Olga Sorkine-Hornung, Daniel Cohen-
Or, and Niloy J. Mitra. 2014. Meta-representation of Shape Families. Transactions on
Graphics (Special issue of SIGGRAPH 2014) (2014), 11 pages. * joint first authors.
Thomas Funkhouser, Michael Kazhdan, Philip Shilane, Patrek Min, William Kaeter,
Ayellet Tal, Szymon Rusinkiewicz, and David Dobkin. 2004. Modeling by example. In
ACM transactions on graphics (TOG) 23, 3 (2004), 652–663.
Rinon Gal, Amit Bermano, Hao Zhang, and Daniel Cohen-Or. 2021. MBGAN: Multi-
Rooted 3D Shape Generation with Unsupervised Part Disentanglement. In ECCV
Workshop on Structural and Compositional Learning on 3D Data (StruCo3D).
Ran Gal, Olga Sorkine, Niloy J Mitra, and Daniel Cohen-Or. 2009. WBRES: An analyze-
and-edit approach to shape manipulation. In ACM SIGGRAPH 2009 papers. 1–10.
Lin Gao, Jie Yang, Tong Wu, Yu-Jie Yuan, Hongbo Fu, Yu-Kun Lai, and Hao Zhang.
2019. SDM-NET: Deep Generative Network for Structured Deformable Mesh. ACM
Trans. Graph. 38, 6, Article 243 (nov 2019), 15 pages. https://doi.org/10.1145/3355089.
3354488.
Kyle Genova, Forrester Cole, Aveneesh Sa, Aaron Sarna, and Thomas Funkhouser. 2020.
Local deep implicit functions for 3d shapes. In Proceedings of the IEEE/CVF Conference
on Computer Vision and Pattern Recognition. 4857–4866.
Kyle Genova, Forrester Cole, Daniel Vlasic, Aaron Sarna, William T Freeman, and
Thomas Funkhouser. 2019a. Learning shape templates with structured implicit
functions. In Proceedings of the IEEE/CVF International Conference on Computer
Vision. 7154–7164.
Kyle Genova, Forrester Cole, Daniel Vlasic, Aaron Sarna, William T Freeman, and
Thomas A Funkhouser. 2019b. Learning Shape Templates With Structured Implicit
Functions. 2019 IEEE/CVF International Conference on Computer Vision (CVF) (2019).
7153–7163.
R. Girshar, D.F. Fouhey, M. Rodriguez, and A. Gupta. 2016. Learning a Predictable and
Generative Vector Representation for Objects. In ECCV.
Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley,
R. Girdhar, D.F. Fouhey, M. Rodriguez, and A. Gupta. 2016. Learning a Predictable and
Generative Vector Representation for Objects. In Conference on Computer Vision
and Pattern Recognition. In Proceedings of the 27th International Conference on
Neural Information Processing Systems - Volume 2 (Montreal, Canada) (NIPS’14). MIT
Press Cambridge, MA, USA, 2672–2680.
Thibault Grousset, Matthew Fisher, Vladimir G. Kim, Bryan Russell, and Mathieu Aubry.
2018. AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation. In
ACM Trans. Graph., Vol. 41, No. 4, Article 106. Publication date: July 2022.
We use the ShapeNet dataset [Chang et al. 2015] for training and we use the train-test split of DeepSDF [Park et al. 2019]. The number of training shapes in each category varies from ∼1K (lamps) to ∼5K (tables).

For each shape we sample points and occupancy labels by the following steps: i) We convert the shape to a watertight mesh using the implementation of Huang et al. [2018]. ii) We scale the shape to a unit sphere. iii) For the GM loss (\(X_{\text{vol}}\) in Eq. (2)), we sample 500, 000 points uniformly inside each shape. iv) For the occupancy loss (\(X_{\text{surf}}\) in Eq. (6)), we randomly sample 500, 000 points inside the bounding cube \([-1, 1]^3\). Additional 500, 000 points are sampled on the surface of the mesh, and are perturbated with Gaussian noise of \(\sigma = 0.01\). 500, 000 more surface points are perturbated with \(\sigma = 0.02\). In total, we sample 1, 500, 000 points for each shape.

For the occupancy labels we use the libigl implementation [Jacobson et al. 2018] of fast winding number [Barill et al. 2018].

Notice that in all the other methods that we compare to [Park et al. 2019; Chen and Zhang 2019; Hao et al. 2020; Genova et al. 2019a], we use the same train/test shapes and use their official implementation\(^2\) for both data preparation and training.

A.2 Network Architecture

Throughout all experiments, our networks were trained using the settings below.

\[Z = 256\]

Shape codes are initialized randomly from \(N(0, 256^{-2})\). Then, as illustrated in Figure 11, the decomposition is similar to PointGMM attentional split layers [Hertz et al. 2020], where, first, a fully connected layer of \(f_0\) splits \(Z\) into \(p = 16\) part vectors of dimension \(512\) each. Afterwards, we apply \(4\) multi-head attention layers with \(h = 4\) number of heads to output the part embedding \(Z^d \in \mathbb{R}^{p \times 512}\).

\[f_m\] and Occupancy Network \(f_o\) are utilized through the full Transformer architecture [Vaswani et al. 2017]. We removed the self-attention layers from the decoder, and used learned positional encoding for the decoder as well. These changes are described in details in Section 3.2.

We use \(d_{\text{model}} = 512\) as the dimension of the Mixing Network (Transformer Encoder). For the Occupancy Network (Transformer Decoder), we set the dimension of \(d_{\text{pe}}\) to \(256\). The Mixing Network, transformer encoder, has \(T = 4\) multi-head attention layers with \(h = 4\) number of heads. The Occupancy Network has \(T = 6\) multi-head attention layers with \(h = 8\) number of heads. To output the occupancy indicator \(\hat{y}\), we process \(\mathbf{x} \in \mathbb{R}^{4d_{\text{pe}}}\) through another MLP with five hidden layers of size 512.

A.3 Training

Each network was trained to minimize the loss term in Eq. (10) for 2000 epochs with batch size of 18 shapes. Each shape in the batch is represented by 2000 uniformly sampled points inside \(X_{\text{vol}}\) and 6000 occupancy points of \(X_{\text{surf}}\), where both are randomly selected from the pre-processed points, as described in A.1.

For the disentanglement loss Eq. (9), the subset \(Z^b\) is composed of \(4 \leq L \leq 12\) part embeddings as described in Section 3.3, where the value of \(L\) is randomly selected in each training iteration.

\(^2\)IM-NET https://github.com/czq142857/IM-NET-pytorch
DeepSDF https://github/facebookresearch/DeepSDF
DualSDF https://github.com/zechunhao995/DualSDF
LIDF https://github.com/google/ldif

---

1http://www.vtk.org.
Fig. 11. Architecture of Decomposition Network $f_d$. Shape embedding $z$ is projected through a fully connected layer to $p$ intermediate part embeddings of dimension 512 each. Then, the embeddings are forwarded through self attention layers to produce output part embedding $z^d \in Z^d$ of 512 dimensions.

We set the loss weight in Eq. (10) to $\gamma = 10^{-4}$. We use the Adam optimizer [Kingma and Ba 2014] with a learning rate of $10^{-4}$ and the default settings ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$). We use 2000 warm-up iterations and apply exponential learning decay of 0.9 in intervals of 500 epochs.

The affine transformations that are applied on $X_{surf}$ and $X_{surf}'$ (see Section 3.3) are randomly selected from 100,000 pre-computed transformations. Each is composed from a random translation $t \in [-0.3, 0.3]^3$, random uniform scale $s \in [0.7, 1.3]$ and a random uniform rotation from $SO(3)$.

Each model was trained on a single RTX A6000 GPU for 1-5 days, depending on the size of the training data. We run our user interface on a laptop equipped with RTX 3700 GPU.

A.4 Shape Inversion Optimization

For shape inversion, we perform a 2-step optimization, as described in Section 3.6. Both the first and second steps of the optimization are run for a fixed amount of 250 iterations.

On a machine equipped with a RTX 3700 GPU, the processing time of 800 shapes takes 2 hours, where shapes are processed in batches of 20 in parallel.

B RANDOM GENERATION RESULTS

From next page we show qualitative comparisons for the random generation evaluation in Section 4.3. For each method and shape category we show the first 36 generated shapes.
Fig. 12. Shape generation results. First 36 sampled airplanes used for the comparison in Table 3.
Fig. 13. Shape generation results. First 36 sampled chairs used for the comparison in Table 3.
Fig. 14. Shape generation results. First 36 sampled tables used for the comparison in Table 3.