Shape Unicode: A Unified Shape Representation

Sanjeev Muralikrishnan[1*] Vladimir G. Kim[1] Matthew Fisher[1] Siddhartha Chaudhuri[1,2]
[1] Adobe Research [2] IIT Bombay

Abstract

3D shapes come in varied representations from a set of points to a set of images, each capturing different aspects of the shape. We propose a unified code for 3D shapes, dubbed Shape Unicode, that imbibes shape cues across these representations into a single code, and a novel framework to learn such a code space for any 3D shape dataset. We discuss this framework as a single go-to training model for any input representation, and demonstrate the effectiveness of the learned code space by applying it directly to common shape analysis tasks – discriminative and generative. In this work, we use three common representations – voxel grids, point clouds and multi-view projections – and combine them into a single code. Note that while we use all three representations at training time, the code can be derived from any single representation during testing. We evaluate this code space on shape retrieval, segmentation and correspondence, and show that the unified code performs better than the individual representations themselves. Additionally, this code space compares quite well to the representation-specific state-of-the-art in these tasks. We also qualitatively discuss linear interpolation between points in this space, by synthesizing from intermediate points.

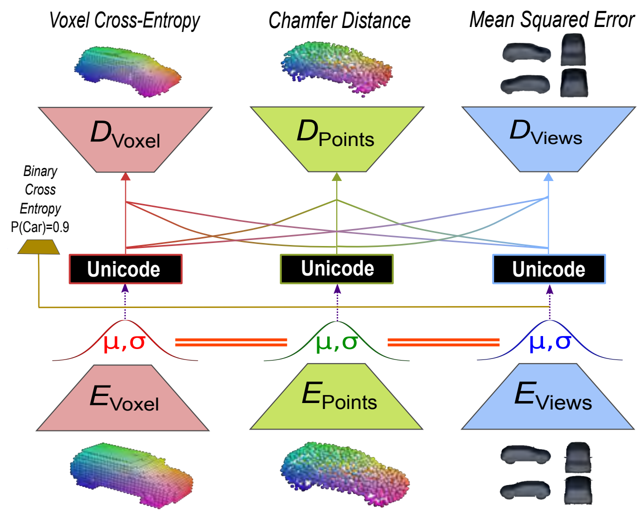

Figure 1: Overview of our Shape Unicode architecture. Different base shape representations (bottom) – voxel grids, point clouds and multi-view images – are processed through three encoder/decoder pairs that are trained to project all three representations to the same unified code (middle). We show that this code is a richly informative input for a range of unified geometry processing pipelines – translation, synthesis, segmentation, correspondences, and retrieval/classification – and can be computed even if only one base representation is available at test time.

1. Introduction

With advances in low-cost sensing devices and 3D authoring tools, repositories of naturally acquired and synthesized 3D shapes have steadily grown. This influx of 3D shape data has brought about advancements in shape analysis and synthesis, and has led to several efficient ways of storing and representing shapes, such as polygon meshes, voxel grids, point clouds, depth maps, projected images, and implicit functions. Each representation is best suited to specific tasks, but none for all such tasks.

As shape data structures, different representations are designed to optimize tasks such as efficient rendering, interactive manipulation, level-of-detail retrieval, and functionality analysis. The advent of deep learning has favoured representations amenable to analysis and generation by neural networks. Further, 3D sensing of real objects added ease-of-acquisition and completeness-of-information to the mix, ranging from single images or depth maps of objects, to full scans from all directions capturing both color and geometry.

Product applications and academic research have chosen shape representations suited for each task and tailored frameworks around them. When applied to deep networks, this has produced modules specialized to each representation, including specialized convolution operations, architectures, loss functions, and training procedures (e.g. meshes [20], point clouds [23], voxels [38], multi-view images [34], octrees [26, 35]). These pipelines are specific to the associated representations and technical innovations developed for one representation rarely carry over to others. Thus, the design cost of analysing data in different

*Corresponding author: samk@adobe.com

representations is proportional to the number of such representations. While translations between representations are possible, this itself is a hard problem. Further, some representations are simply less suited for a given task (e.g. if they lack specific information important for that task, such as high-resolution detail), yet may be the most natural form in which shapes can be acquired.

In this work, we address these two challenges — pipeline multiplicity and differential performance — by proposing a single, unified, non-category-specific encoding of 3D shapes that can be generated from any of a variety of base representations. Unified analysis pipelines can then be trained on this common code space. We show that such analyses benefit from the collective strengths of different representations injected into the code during training: they outperform pipelines trained on individual representations, and promote consistent performance regardless of the input representation. Further, the encoding is invertible, and can be used for translation and generation.

Our work is inspired by Hegde et al. [10], who train a network that processes two representations of a single shape (voxel grids and multi-view images) in parallel, and combine the final classification predictions with an additional layer. Each branch picks up cues best captured in the associated representation, improving performance of the overall ensemble. Similarly, Su et al. [32] combine point clouds and RGB images to better segment facades and shapes. In contrast to these works, our method does not assume multiple representations for a shape are available at test time: thus, it is not an ensemble-based approach. Instead, a shape can come in *any* supported representation, and still be projected accurately to the common code space. The strengths of other representations are "hallucinated" in the construction of this code, because of the training process.

We learn such a code space by jointly training encoders for different representations to converge to the same high-dimensional code, that is then decoded by a decoder for each representation. Translation losses on the decoded output, in addition to direct similarity losses on the codes, ensure that the learned code imbibes salient information from each representation. We then perform shape classification, segmentation, and dense correspondences by training task-specific but representation-independent neural networks purely on top of the learned code. Although our method can be used with any representation, for this paper, we chose three common input representations – voxels, point clouds and multiview projections.

Our main contribution is a unified encoding of a 3D shape learned from a variety of base representations, and we demonstrate that this encoding is:

- more informative than one learned from a single representation, yet
- computed from a *single* representation at test time, and
- useful in a wide range of applications, such as classification, retrieval, shape generation, segmentation, and correspondence estimation.

This offers a representation-invariant framework that performs consistently well on different tasks, even if representations at training and testing time are different.

2. Related Work

We overview common representations used for shape analysis, as well as recent methods that explore combining these representations.

Shape Representations for 3D Learning. Unlike images, 3D shape representations, such as meshes, lack a common parametrization, which makes it hard to directly extend 2D deep learning methods to 3D. Early approaches convert input shapes to 3D voxel grids which can be used with natural extensions of 2D convolutional networks. Existing methods tackle classification [38], segmentation [22], registration [40], and generation [37]. Since the shape surface usually occupies a small fraction of the voxels, various extensions leverage data sparsity by directly processing more efficient data structures such as octrees [26, 35, 36].

While voxel grids provide a natural parameterization of the domain for learning convolutional filters, they traditionally struggle to capture finer details. Multi-view shape analysis techniques demonstrate that many common problems can be addressed by using 2D renderings or projections of a 3D model as the input for a neural network. These representations enabled new architectures for standard tasks: shape classification [34, 13], segmentation [12], and correspondences [11]. Surface parameterization techniques can also be used instead of projections to map shapes to images [19, 30, 31, 9]. Image-based shape analysis methods enable capturing finer details in higher resolution images. However, they are typically memory intensive since many viewpoints need to be captured for holistic shape understanding, and they are not ideal for 3D synthesis and design tasks since the native 3D shape must be separately reconstructed from its image collection.

Surface-based models have been proposed to directly analyze point clouds [23, 25], meshes [17], and general Riemannian surfaces [6, 20, 2]. Point-based methods are convenient for data preparation, and offer a compact representation. Hence, they are popular choices for common shape analysis tasks [5, 4]. However, representing geometric details or learning convolutional filters to detect finer features is typically difficult with existing point architectures.

Translation between representations is possible: e.g. Girdhar et al. [7] chain encoders and decoders to map images to their corresponding 3D shapes. Su et al. [33] use 3D shape similarity to construct an embedding space and learn to map images to that space, enabling cross-domain retrieval. The limitation of these translation techniques is that their low-dimensional shape codes are derived from a single representation, and thus the embedding does not take into account the additional information that might be available in alternative representations.

Hybrid representations. Several methods leverage the complementary nature of various representations. For example, Atzmon et al. [1] map point functions to volumetric functions to enable learning convolutional filters, and then project the learned signal back to points. The converse is also possible: one can subdivide large point clouds using a coarse voxel grid, and then use point-based analysis within each voxel to learn per-voxel features used in a voxel CNN [21]. Volumetric CNNs can take advantage of higher-resolution filters akin to multi-view networks by learning anisotropic probing kernels that sample more coarsely along particular dimensions [24]. Features obtained from images can be used as input to point-based architectures [39]. One can treat color information jointly with point coordinates by representing them as sparse samples in a high-dimensional space, a datastructure efficiently analyzed with sparse bilateral convolutional layers [32]. These approaches require deriving novel network layers, and are usually suited only for some tasks. A more general meta-technique simply aggregates features derived from multiple representations [10, 29] or at multiple scales [41, 18].

All such methods require all representations to be available during testing. Instead, we show that one can map *any* individual representation available at test time to a rich, informative code on which discriminative or generative models can be trained, with consistent performance regardless of input representation. Existing hybrid techniques also focus on one task, such as image-based shape generation or classification. We demonstrate that our shape code trained with cross-representation reconstruction loss extends well to a variety of tasks, such as correspondence estimation and segmentation, in addition to classification and generation.

3. Method

Our *Shape Unicode* model is based on an autoencoder architecture, where encoders and decoders are defined for all possible representations (see Figure 1). At a high level, our method is designed around two central principles:

- Each representation's code should incorporate information from all input representations.
- At test time, we should be able to obtain this shape code from *any* input representation, in the absence of the other representations.

To achieve these goals, we add an additional loss function that favors the codes for the same shape obtained from different representations to be the same. During training, we feed the output of each encoder through each of the three possible decoders, forcing any decoder to reconstruct the shape even if it was encoded from a different representation. These two losses ensure that our principles are satisfied, and that the code is as-informative-as-possible for reconstruction. We train a single network for all shape categories, and thus to favor their separability in the code space, we also build a small layer that maps codes to classes and add a classification loss.

We test our approach with three commonly-used input shape representations: voxel binary-occupancy grids, point clouds, and multi-view images from four views (their branches are colored red, green, and blue, respectively in Figure 1). We represent a voxel grid as 32^3 occupancy values, a point cloud as 1024 XYZ points sampled from the object surface, and use four grayscale 128x128 images for our multi-view representation.

In the remainder of this section we provide details for our encoders and decoders (Sec. 3.1), loss functions (Sec. 3.2), and training procedure (Sec. 3.3).

3.1. Unicode Architecture

Our design for individual encoders and decoders is motivated by the variational auto-encoder (VAE) approach [16]. That is, for each representation, the respective encoder predicts the mean and standard deviation of a Gaussian distribution describing a small neighborhood of codes that map to the input shape. We use the re-parameterization trick [16] to sample a 1024-dimensional code vector from the Gaussian, and the decoder has to reconstruct the same target from the sample. This approach forces the decoder to be more robust to small code perturbations, and usually converges and generalizes better. Note, however, that we do not impose the overall distributional constraint of VAEs, since we found it over-constrained our optimization and led to cluster overlap (and hence poorer class discrimination) in the embedding space.

At training time we translate between all pairs of input/output representations. Each code derived from each representation is fed into each decoder providing nine encoder-decoder pairs (a cross product of three encoders and three decoders). Each code is expected to both reconstruct the shape in its source representation and to translate to an instance of the shape in each output representation, encouraging the code to capture information across all representations.

See the supplementary materials for a detailed description of the architecture of the encoder and decoder networks for each of the representations.

3.2. Unicode Loss Functions

Here we detail the training losses we use to guide our joint Unicode network. Let $\mathcal{R}$ denote the set of possible representations: $\mathcal{R} = \{Voxel, PointCloud, MultiView\}$.

Reconstruction Loss The reconstruction loss encourages each decoder output to match the input for each representation:

$$\mathcal{L}_{reconstruct} = \sum_{x \in \mathcal{R}} \sum_{y \in \mathcal{R}} w_{x \to y} \mathrm{Dist}_y(D_y(E_x(S_x)), S_y), \quad (1)$$

where for each pair of input (x) and target (y) representations, the shape S is encoded from its input representation $E_x(S_x)$ and then decoded into target representation via D_y. We then compare the decoded shape to the ground truth shape S_y in the target representation. The distance functions Dist_y are used to compare reconstructed and true target shape, and thus have to be defined differently for each representation. We use per-voxel cross-entropy for voxels, Chamfer distance for point clouds [5], and mean squared difference for multi-view images. The terms where $x \neq y$ are also referred to as *translation losses*.

Embedding Loss The embedding loss encourages the embeddings generated by each encoder to be similar. The output of each encoder E_x is a mean and standard deviation, which we constrain via L_1 loss:

$$\mathcal{L}_{embedding} = \sum_{x \in \mathcal{R}} \sum_{y \in (\mathcal{R} \setminus x)} | E_x - E_y |_1 \quad (2)$$

Classification Loss To further aid discriminative tasks with the learned code, we encourage the embedding space to be structured such that different classes of shapes map to different, well-separated clusters. This is accomplished by adding a shared single fully connected layer that uses the code derived from any of the representations as input. Note that we use only a single set of weights for this layer, which is *shared* across all representations. $\mathcal{L}_{classification}$ is the resulting cross-entropy classification loss summed across the output of each encoder.

Total Loss For a single input shape, the total loss is simply a weighted sum of the above losses:

$$\mathcal{L}_{total} = \mathcal{L}_{reconstruct} + \mathcal{L}_{embedding} + \mathcal{L}_{classification} \quad (3)$$

This loss is then averaged over the mini-batch of shapes.

3.3. Unicode Training

Adaptive Loss Weighting For each pair of representations used to compute $\mathcal{L}_{reconstruction}$, we prescribe a weight parameter $w_{x \to y}$ to scale the loss term. This is done because different representation losses (chamfer distance, binary cross-entropy, and MSE) produce values that are a few orders of magnitude apart. At the start of each epoch (i.e., a complete run over the whole dataset) we compute these weights and keep them fixed for the epoch. Specifically, we pick a random mini-batch and compute weights $w_{x \to y}$ s.t. $w_{x \to y} \times L_{x \to y} = L_{max}$, where L_{max} is the *maximum* loss value among those 9 terms for that mini-batch. The chosen mini-batch for adaptive weighting is selected at random before training begins, and is unchanged thereafter.

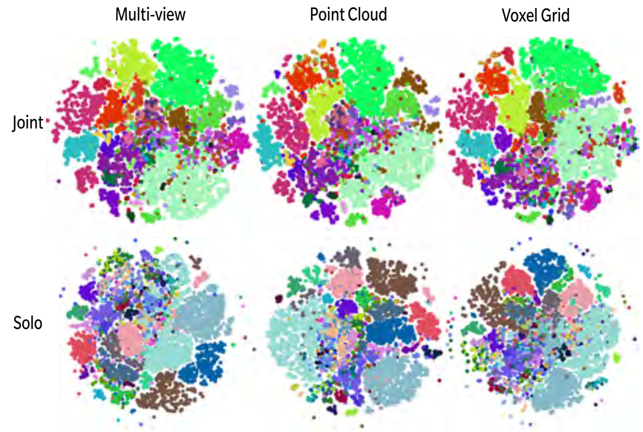

Figure 2: *Top:* t-SNE plots of each input representation's projection to the Shape Unicode space (for consistency, we ran t-SNE on the union of embeddings, and visualized each embedding using the same axes). Our joint code space, driven by pairwise code differences, maps inputs from multiple representations to similar points. Points from the same class have the same color. *Bottom:* Codes trained with three independent autoencoders lack similar consistency.

Training Parameters Our joint code space is 1024-dimensional. We feed all 3 representations for each shape in a batch. We use a mini-batch of 16 shapes, with *Adam* optimizer [15] and a learning rate of 0.001 during both our joint training and subsequent task-level training , with the exception of the dense correspondence task which uses a learning rate of 0.0001. More architecture and training details are illustrated in the supplementary material.

4. Results

In this section, we present different evaluations of our unified encoding of shape representations. First, we describe the training datasets used for all our experiments. Then, we qualitatively study the embeddings produced by the three representation encoders, and show that a single code is successfully invertible to any of the source representations. We also show that the codes form a reasonably compact space that allows interpolation between shapes, with synthesis of plausible, novel intermediates, regardless of input representation. Finally, we present quantitative comparisons against several ablations of our method, using classification accuracy as the evaluation metric. We also compare against alternative architectures that use multi-representation ensembles [10] and code concatenation. In the next section, we present further evaluations of our method in the context of applications to standard shape analysis tasks (segmentation, correspondences, and retrieval), and show that the jointly trained encoding leads to measurable improvements in all these tasks.

Datasets. We use the ShapeNet dataset [3], using 35763 shapes for training, 5133 for validation, and 10265 for testing, following the split used in prior work [34]. The shapes are categorized into 55 classes that we use in our classification loss (the code is not category-specific). Original models are stored as polygon meshes, and we use standard tools to derive all three representations – voxel grids, point clouds, and multi-view images – from each mesh.

Embedding similarity. Having a joint code space where multiple representations map to the same code allows us to have a unified training pipeline for all applications that the code may be used for. Given a common output representation, our application-specific models trained over this code space are thus shared across representations. In addition, any test-time representation can now account for the missing representations by hallucinating their code. This eliminates the need to spend time searching for a perfect representation for a task. It also makes it possible to train when ground truth comes in a mix of representations (e.g., if training data is compiled from multiple sources), or if representation at test time differs from training data.

Figure 2 shows a t-SNE plot of the embedding space as computed on the ShapeNet Test dataset of 10265 shapes, for each of the three input representations. Each 2D point in the plot is colored according to one of 55 class indicators. We observe that the class-level clusters are well-formed and separated from other classes, while the embeddings remain similar for all three representations (top row, **joint**). In contrast, codes trained individually per input representation (i.e. training each encoder/decoder pair individually) do not have this consistency (bottom row, **solo**).

Translation and Reconstruction. By training our model in a reconstruction/translation setting, we ensure that each representation's code preserves geometric cues from all representations. In addition to making the code space rich in information, our trained model can be directly used for inter-representation translations. While some translations are simple, such as 3D to 2D representations or fine to coarse representations, our model also provides non-trivial *Voxel→Point Cloud*, *Multi-View→Point Cloud* and *Multi-View→Voxel* translations. Figure 3 shows some sample reconstruction and translation results (see supplementary materials for additional examples). One trend we observed is that translations *to* voxels are of a marginally lower quality than translations to other representations; a precursor to the fact that voxels benefit most from joint training, as shown in later discussions.

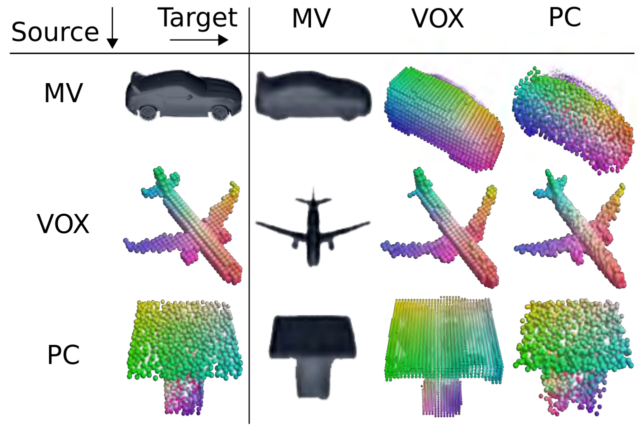

Figure 3: Shape reconstruction and translations between all three representations, for three test shapes.

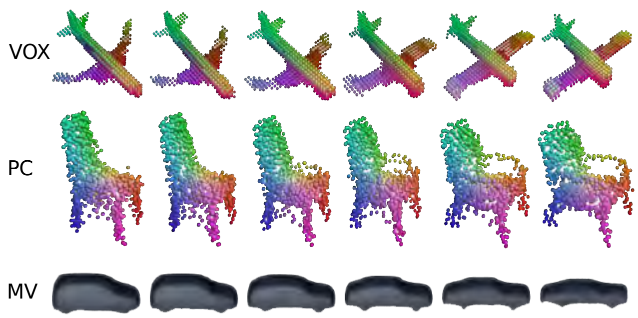

Figure 4: Shape Generation: We linearly interpolate between a source shape's code *(Leftmost)* and a target shape's code *(Rightmost)*, and decode the intermediate novel codes to chosen representations. Decodings to the other two representations are shown in supplementary material.

Interpolation and Generation. Even though our model is only halfway to a variational auto-encoder – it does not impose a global distribution constraint – the various losses induce it to learn a fairly compact distribution, yet with well-separated classes. As a result, we can attempt some shape interpolation tasks in this space, by linearly interpolating between the codes of source and target shapes, and decoding the intermediate codes to any desired representation. This is quite successful: we show three examples in Figure 4, for three different representations. In the supplementary material, we show that each intermediate code in the figure can also successfully generate the two remaining representations. We observe that these novel codes smoothly effect both geometric and topological changes such as generating chair arms.

Ablations. We show that different ablations of our joint training model result in reduced performance. As a canonical task for these ablations, we choose classification of the 10265 test shapes into the 55 ShapeNet classes. After training the encoders/decoders, we freeze them and train a simple classifier on top of them (3 fully-connected layers of size 512, 256 and 55, + softmax). The classifier shares weights across all input representations since it operates on a unified code space, *except* for the first two ablations below, where the codes are not expected to be similar across representations and sharing weights would be an unfair disadvantage. The ablations we perform are:

Solo Training*: The encoder/decoder pair for each input representation is trained independently, without any cross-representational losses.

Without Embedding Loss*: Joint training without the L_1 loss forcing codes generated by different encoders for the same shape to be identical.

Without Translation Loss: Joint training without translation losses (reconstruction losses when encoder and decoder are for different representations).

Without Classification Loss: Joint training without the classification loss.

(* 3 independent classifiers)

The results are in Table 1, for each input representation (multi-view, point cloud, or voxel grid) of ShapeNet test shapes. It can be observed that joint training with all the losses produces the most informative code under this metric, outperforming independently trained autoencoders even when the latter have dedicated classifiers. A **key takeaway** from this table is that Shape Unicode disproportionately improves classification of shapes input as voxel grids, by 1.72%. The performance for multi-view and point cloud input, which were already quite high with solo training, does not change much. However, the underperforming representation gets a free boost by being forced to mimic codes from the high-performing representations. It is important to note, that only one representation is given to the network at test time, so a better code is derived from a weak representation with our method. We will see this pattern recur in our shape retrieval experiments.

	Multi-View	Point Cloud	Voxels
Shape Unicode	83.38	**84.23**	**82.48**
Solo Training*	**83.53**	84.07	80.76
W/o Embedding*	81.61	81.77	81.11
W/o Translation	81.72	82.36	79.01
W/o Classification	81.91	82.28	81.67

Table 1: ShapeNet classification accuracy, for our method vs ablations with different loss terms turned off. Solo training means all joint, cross-representational losses are turned off and the individual autoencoders are trained independently. The simple classifier, trained after freezing the codes, shares weights across all three input representations *except* in the first two ablations (marked with *), where we train three independent classifiers so as not to unfairly disadvantage them.

	Multi-View	Point Cloud	Voxels
Shape Unicode	83.38	84.23	82.48
Weighted Fusion		81.54	
Code Concatenation		81.53	

Table 2: Comparison with code fusion alternatives, on ShapeNet classification. Since fusion combines all 3 representations into a single prediction, only one value is computed for these rows.

Code Fusion Alternatives. We further test existing strategies for using ensembles of multiple representations. These techniques assume all of the latter are available at test time, while our method needs only one representation for testing. *Weighted fusion* [10] combines predictions based on several representations by taking a weighted average of the classification probabilities, with learned weights. *Code concatenation* [21, 38, 29] concatenates the codes output by each representation's model and layers a predictor on top of the combined code. We re-used our encoder/decoder architectures in the above ensemble strategies, so that base architectural differences were not a factor. The ShapeNet classification results are shown in Table 2. Our model outperforms both code fusion strategies by a significant margin, while still employing a single *shared* classification network and without needing all representations during prediction. We found that adaptive loss weighting plays an important role when combining multiple representations, since the losses are highly asymmetric in magnitude.

5. Applications

In this section, we present unified pipelines built on top of Shape Unicode for three fundamental shape analysis tasks: segmentation and labeling, dense correspondences, and retrieval. For each task, we build a framework to ingest codes generated from any representation, and process them identically. Note that the encoders producing the code are frozen for these tasks and not further tuned: the pipelines operate on a single, standard code. Our goal is to show that:

1. Even with relatively simple encoder/decoder architectures and coarse input representations (just 32^3 voxels, 1024 points, 4 views), our pipelines compare

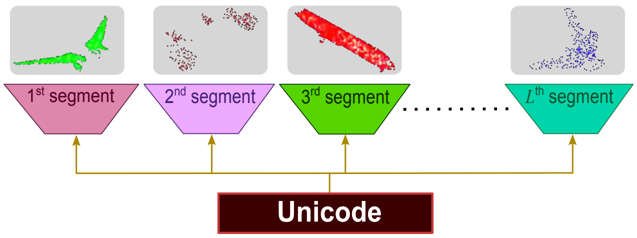

Figure 5: Shape segmentation architecture. Representation-agnostic, per-label decoders map an input unicode to point clouds for shape parts.

quite well with the state of the art and are entirely representation-agnostic.

2. The jointly trained code is enriched by multiple representations during training, enabling the unified pipelines to outperform comparable architectures separately developed for each individual representation.
3. The use of our common code equalizes performance across input representations for all tasks.

5.1. Shape Segmentation

We perform shape segmentation on ShapeNet, and compare our accuracy to state-of-the-art methods. The ground truth is labeled point clouds. Prior methods have found ways to compare voxel, mesh or multi-view solutions to these points (e.g. through surface projections). However, with Shape Unicode we can directly map any unsegmented input representation to a segmented point cloud, using a single segmenting decoder that ingests the common code.

Since our shape code describes the entire shape, we need to derive per-point labels from it. We pass the pretrained shape code (regardless of which encoder produced it) through L part decoders with identical architecture, where L is the number of ground truth part labels in the shape class (Figure 5). Each part decoder is a series of fully connected layers mapping the code to 3025 points describing the part. We train the decoders with a Chamfer distance loss [5] w.r.t. the ground truth segments. Since these have varying cardinalities, we map the outputs to the query points with nearest neighbor lookup. As in prior methods, we train independently for each shape category in this experiment.

	Average Segmentation Accuracy		
WUNet [22]	90.24		
ShapePFCN [12]	89.00		
Joint	86.73	87.26	86.79
Solo	85.73	85.05	–
Input $\rightarrow$	Point Cloud	Voxels	Multi-View

Table 3: Overall ShapeNet segmentation accuracy.

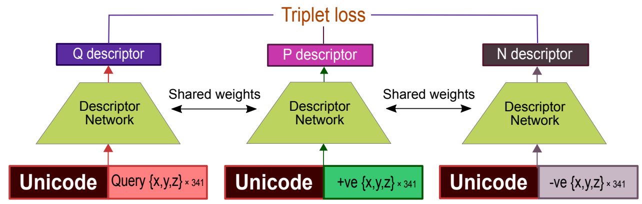

Figure 6: Shape correspondence training setup. Two corresponding points and one mismatched point, whose coordinates are concatanated with the unicodes of their parent shapes, are independently mapped to descriptors by a network, followed by a triplet loss.

We report segmentation accuracy in Table 3. Our results ("Joint" row) compare quite well with the state of the art, and are consistent enough to be completely input representation-agnostic. While we undoubtedly exploit consistent alignment of shapes, we did not spend much time optimizing the decoders or using specialized layers like CRF. As an ablation, we show results generated from "solo" codes trained on individual representations. For point clouds, we use the same setup. For voxel grids, we do not assume we can freely translate it to a different representation. Instead, we use part decoders that each output a 32^3 voxel grid for the part. The labeled voxel centroids are compared to the GT point cloud. For multi-view, we cannot trivially compute per-point accuracy since the 2D$\rightarrow$3D projection is hard: hence we omit it. Our unicode approach makes it possible to accommodate input representations like multi-view in a single, consistent, training and testing framework. The jointly trained codes yield a 1-2% improvement over solo codes in case where both exist. Per-class accuracies are provided in supplementary material.

5.2. Dense Shape Correspondence

In this experiment, we use the shape code for estimating dense point correspondences. Again, since our code does not provide precise per-point information, we need a new network for correspondence evaluation. This network ingests the x, y, z coordinates of a point on the shape (repeated several times to fix dimensional imbalance), along with the unicode, and outputs a 16-D point descriptor (Figure 6). This point descriptor can later be used to compare points across shapes and form correspondences between them. We learn the point descriptors using triplets of two corresponding and one non-corresponding points [28], selected using Semi-Hard Negative Mining. We do this for voxels and point clouds inputs, since the ground truth is in the form of point sets. For point cloud inputs, the x, y, z are simply point coordinates, while for voxels they are the indices of the grid cell that contains the point. We train this model using approximate correspondences obtained with non-rigid alignment on ShapeNet [11].

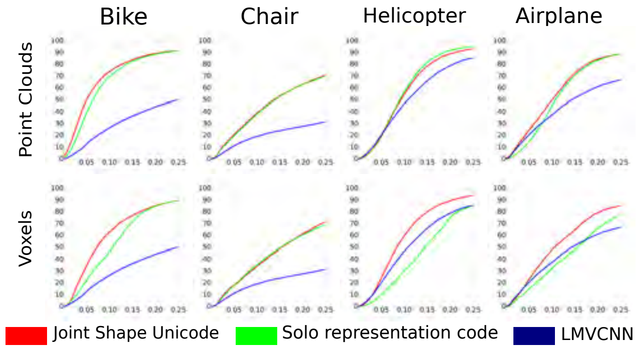

Figure 7: *Dense correspondence:* Euclidean distance from ground-truth point (x-axis) vs Accuracy (y-axis) with Shape Unicode, solo code and LMVCNN on the BHCP dataset [14]. Note that LMVCNN is rotation-invariant, and does not leverage the fact that the dataset is aligned. It is thus provided only as a reference.

We test on the BHCP benchmark [14], which contains 100 shapes each of Airplanes, Bikes, Chairs and Helicopters with manually annotated keypoints. We train the network to extract point descriptors independently for each category. Note that our training data excludes helicopters, hence we use the model trained on airplanes in this case [11]. Since our unicode model was trained on the aligned+normalized ShapeNet dataset and isn't rotation-invariant, we align and normalize the BHCP shapes similarly to ShapeNet shapes.

To evaluate our results, we use standard criteria, reporting the fraction of points falling within the ground truth correspondence at each distance threshold (Figure 7). Note that our method performs consistently well with either representation as input, when the code is trained jointly (red), while results degrade for codes trained on only one representation (green). The comparable method of LMVCNN [11] performs even worse, but this is an unfair comparison, provided only for reference, since LMVCNN is rotation-invariant while Shape Unicode is not. Also note that our model, like MVCNN, generalizes to the unseen helicopter class.

5.3. Shape Retrieval

We perform shape retrieval using our joint code and evaluate it against methods presented as part of the SHREC'17 retrieval challenge [27]. We use our 3-layer classification network described in Section 4. Retrieval is then performed for a query shape by selecting other shapes of the same *predicted* class as the query shape. This selection is then sorted by confidence of class prediction indicated by the output class score. We evaluate using F1-score as implemented in benchmark software [27] and present the results in Table 4. We compute both *micro* and *macro* averaged versions of this metric, with the former accounting for class population sizes and the latter without any weighting. The results

	Micro F1	Macro F1
RotationNet	0.80	0.59
GIFT	0.77	0.58
ReVGG	0.77	0.52
MVCNN	0.76	0.58
PC Joint	0.73	0.50
PC Solo	0.73	0.49
MV Joint	0.72	0.48
MV Solo	0.72	0.48
DLAN	0.71	0.51
Vox Joint	0.71	0.48
MV FusionNet	0.69	0.48
Vox Solo	0.68	0.46
CMCNN	0.48	0.17
ZFDR	0.28	0.20
VoxelNet	0.25	0.26

Table 4: Shape retrieval comparison against methods in SHREC'17 [27]

in Table 4 are sorted by *Micro F1*, with ties resolved by *Macro F1*. Our approach (**Joint**) achieves comparable results to other methods, with any input representation. As in other cases, training each representation independently (**Solo**) noticeably damages performance on voxel grids.

6. Conclusion

We have presented a framework for generating a joint latent space for 3D shapes that can be encoded from any input representation, and either decoded to another representation or used directly for tasks such as shape classification, retrieval, correspondence estimation, or segmentation. We demonstrate that a code derived from multiple representations is more informative and leads to better quantitative results even if only one representation is available at test time. Future techniques can build on our framework to create representation-invariant methods. This would reduce the time spend searching for a perfect representation for a task, would facilitate training when ground truth comes in a mix of representations (which often happens when it's borrowed from multiple sources), and also help when the representation at test time is different from the training data.

Although we have only explored voxel, point cloud, and multi-view renderings in this work, it is natural to use additional representations such as atlases [8], patch-based octrees [36], or surfaces [17] to further increase the representative power and generalizability of the latent code. Our core architecture is also developed for encoding the entire shape. While we provide a way to decode this into more detailed per-point signals, it would be interesting to create tools for deriving a universal code for on-surface features that could better capture fine-scale geometric detail.

References

[1] M. Atzmon, H. Maron, and Y. Lipman. Point convolutional neural networks by extension operators. *ACM Trans. Graph.*, 37(4):71:1–71:12, 2018.

[2] D. Boscaini, J. Masci, E. Rodoià, and M. Bronstein. Learning shape correspondence with anisotropic convolutional neural networks. In *NIPS*, 2016.

[3] A. X. Chang, T. A. Funkhouser, L. J. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: An information-rich 3D model repository. *CoRR*, abs/1512.03012, 2015.

[4] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner. ScanNet: Richly-annotated 3D reconstructions of indoor scenes. In *CVPR*, 2017.

[5] H. Fan, H. Su, and L. J. Guibas. A point set generation network for 3D object reconstruction from a single image. In *CVPR*, 2017.

[6] M. Fey, J. E. Lenssen, F. Weichert, and H. Müller. SplineCNN: Fast geometric deep learning with continuous B-spline kernels. In *CVPR*, 2018.

[7] R. Girdhar, D. F. Fouhey, M. Rodriguez, and A. Gupta. Learning a predictable and generative vector representation for objects. In *ECCV*, 2016.

[8] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry. Atlasnet: A papier-mâché approach to learning 3d surface generation. *CVPR*, 2018.

[9] H. B. Hamu, H. Maron, I. Kezurer, G. Avineri, and Y. Lipman. Multi-chart generative surface modeling. In *SIGGRAPH Asia*, 2018.

[10] V. Hegde and R. B. Zadeh. FusionNet: 3D object classification using multiple data representations. *CoRR*, abs/1607.05695, 2016.

[11] H. Huang, E. Kalogerakis, S. Chaudhuri, D. Ceylan, V. G. Kim, and E. Yumer. Learning local shape descriptors from part correspondences with multiview convolutional networks. *ACM Trans. Graph.*, 37(1), 2018.

[12] E. Kalogerakis, M. Averkiou, S. Maji, and S. Chaudhuri. 3D shape segmentation with projective convolutional networks. In *CVPR*, 2017.

[13] A. Kanezaki. RotationNet: Learning object classification using unsupervised viewpoint estimation. In *CVPR*, 2018.

[14] V. G. Kim, W. Li, N. J. Mitra, S. Chaudhuri, S. DiVerdi, and T. Funkhouser. Learning part-based templates from large collections of 3D shapes. In *SIGGRAPH*, 2013.

[15] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[16] D. P. Kingma and M. Welling. Auto-encoding variational Bayes. In *ICLR*, 2014.

[17] I. Kostrikov, Z. Jiang, D. Panozzo, D. Zorin, and J. Bruna. Surface networks. In *CVPR*, 2018.

[18] J. Li, B. M. Chen, and G. Hee Lee. SO-Net: Self-organizing network for point cloud analysis. In *CVPR*, 2018.

[19] H. Maron, M. Galun, N. Aigerman, M. Trope, N. Dym, E. Yumer, V. G. Kim, and Y. Lipman. Convolutional neural networks on surfaces via seamless toric covers. In *SIGGRAPH*, 2017.

[20] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst. Geodesic convolutional neural networks on Riemannian manifolds. In *ICCV Workshops*, 2015.

[21] D. Maturana and S. Scherer. VoxNet: A 3D convolutional neural network for real-time object recognition. In *IROS*, 2015.

[22] S. Muralikrishnan, V. G. Kim, and S. Chaudhuri. Tags2Parts: Discovering semantic regions from shape tags. In *CVPR*, 2018.

[23] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. In *CVPR*, 2017.

[24] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. Guibas. Volumetric and multi-view CNNs for object classification on 3D data. In *CVPR*, 2016.

[25] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. PointNet++: Deep hierarchical feature learning on point sets in a metric space. In *NIPS*, 2017.

[26] G. Riegler, A. O. Ulusoy, and A. Geiger. OctNet: Learning deep 3D representations at high resolutions. In *CVPR*, 2017.

[27] M. Savva, F. Yu, H. Su, A. Kanezaki, T. Furuya, R. Ohbuchi, Z. Zhou, R. Yu, S. Bai, X. Bai, M. Aono, A. Tatsuma, S. Thermos, A. Axenopoulos, G. T. Papadopoulos, P. Daras, X. Deng, Z. Lian, B. Li, H. Johan, Y. Lu, and S. Mk. Large-scale 3D shape retrieval from ShapeNet Core55. In *Eurographics Workshop on 3D Object Retrieval*, 2017.

[28] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. *CoRR*, 2015.

[29] D. Shin, C. C. Fowlkes, and D. Hoiem. Pixels, voxels, and views: A study of shape representations for single view 3D object shape prediction. In *CVPR*, 2018.

[30] A. Sinha, J. Bai, and K. Ramani. Deep learning 3d shape surfaces using geometry images. In *CVPR*, 2016.

[31] A. Sinha, A. Unmesh, Q. Huang, and K. Ramani. SurfNet: Generating 3D shape surfaces using deep residual networks. In *CVPR*, 2017.

[32] H. Su, V. Jampani, D. Sun, S. Maji, E. Kalogerakis, M.-H. Yang, and J. Kautz. SPLATNet: Sparse lattice networks for point cloud processing. In *CVPR*, 2018.

[33] H. Su, Y. Li, C. Qi, N. Fish, D. Cohen-Or, and L. Guibas. Joint embeddings of shapes and images via CNN image purification. In *SIGGRAPH Asia*, 2015.

[34] H. Su, S. Maji, E. Kalogerakis, and E. G. Learned-Miller. Multi-view convolutional neural networks for 3D shape recognition. In *ICCV*, 2015.

[35] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong. O-CNN: Octree-based convolutional neural networks for 3D shape analysis. In *SIGGRAPH*, 2017.

[36] P.-S. Wang, C.-Y. Sun, Y. Liu, and X. Tong. Adaptive O-CNN: A patch-based deep representation of 3D shapes. In *SIGGRAPH Asia*, 2018.

[37] J. Wu, C. Zhang, T. Xue, W. T. Freeman, and J. B. Tenenbaum. Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling. In *NIPS*, 2016.

[38] Z. Wu, S. Song, A. Khosla, L. Zhang, X. Tang, and J. Xiao. 3D ShapeNets: A deep representation for volumetric shapes. In *CVPR*, 2015.

[39] D. Xu, D. Anguelov, and A. Jain. PointFusion: Deep sensor fusion for 3D bounding box estimation. In *CVPR*, 2018.

[40] A. Zeng, S. Song, M. Nießner, M. Fisher, J. Xiao, and T. Funkhouser. 3DMatch: Learning local geometric descriptors from RGB-D reconstructions. In *CVPR*, 2017.

[41] Z. Zhu, X. Wang, S. Bai, C. Yao, and X. Bai. Deep learning representation using autoencoder for 3D shape retrieval. *Neurocomputing*, 204:41–50, 2016.