Deep Parametric Shape Predictions using Distance Fields

Dmitriy Smirnov\textsuperscript{1} Matthew Fisher\textsuperscript{2} Vladimir G. Kim\textsuperscript{2} Richard Zhang\textsuperscript{2} Justin Solomon\textsuperscript{1}
\textsuperscript{1}Massachusetts Institute of Technology \textsuperscript{2}Adobe Research

Abstract

Many tasks in graphics and vision demand machinery for converting shapes into consistent representations with sparse sets of parameters; these representations facilitate rendering, editing, and storage. When the source data is noisy or ambiguous, however, artists and engineers often manually construct such representations, a tedious and potentially time-consuming process. While advances in deep learning have been successfully applied to noisy geometric data, the task of generating parametric shapes has so far been difficult for these methods. Hence, we propose a new framework for predicting parametric shape primitives using deep learning. We use distance fields to transition between shape parameters like control points and input data on a pixel grid. We demonstrate efficacy on 2D and 3D tasks, including font vectorization and surface abstraction.

1. Introduction

The creation, modification, and rendering of parametric shapes, such as in vector graphics, is a fundamental problem of interest to engineers, artists, animators, and designers. Such representations offer distinct advantages. By expressing shapes as collections of primitives, we can easily apply transformations and render at arbitrary resolution while storing only a sparse representation. Moreover, generating parametric representations that are consistent across inputs enables us to learn common underlying structure and estimate correspondences between shapes, facilitating tools for retrieval, exploration, style/structure transfer, and so on.

It is often useful to generate parametric models from data that do not directly correspond to the target geometry and contain imperfections or missing parts. This can be an artifact of noise, corruption, or human-generated input; often, an artist intends to create a precise geometric object but produces one that is “sketchy” and ambiguous. Hence, we turn to machine learning methods, which have shown success in inferring structure from noisy data.

Convolutional neural networks (CNNs) achieve state-of-the-art results in vision tasks such as image classification \cite{22}, segmentation \cite{25}, and image-to-image translation \cite{19}. CNNs, however, operate on raster representations.
method over Chamfer distance in Figure 1.

We apply our new framework in 2D to a diverse dataset of fonts, training a network that takes in a raster image of a glyph and outputs a collection of Bézier curves. This effectively maps glyphs onto a common set of parameters that can be traversed intuitively. We use this embedding for font exploration and retrieval, correspondence, and interpolation in a completely self-supervised setting, without need for human labelling or annotation.

We also show that our approach works in 3D. With surface primitives in place of curves, we perform abstraction on ShapeNet [7], outputting parametric primitives to approximate each input. Our method can produce consistent shape segmentations, outperforming state-of-the-art deep cuboid fitting of Tulsiani et al. [41] on semantic segmentation.

**Contributions.** We present a technique for predicting parametric shapes from 2D and 3D raster data, including:

- a general distance field loss function motivating several self-supervised losses based on a common formulation;
- application to 2D font glyph vectorization, with application to correspondence, exploration, retrieval, and repair;
- application to 3D surface abstraction, with results for different primitives and constructive solid geometry (CSG) as well as application to segmentation.

2. Related Work

**Deep shape reconstruction.** Reconstructing geometry from one or more viewpoints is crucial in applications like robotics and autonomous driving [13, 35, 38]. Recent deep networks can produce point clouds or voxel occupancy grids given a single image [12, 8], but their output suffers from fixed resolution.

Learning signed distance fields defined on a voxel grid [9, 37] or directly [30] allows high-resolution rendering but requires surface extraction; this representation is neither sparse nor modular. Liao et al. address the rendering issue by incorporating marching cubes into a differentiable pipeline, but the lack of sparsity remains problematic, and predicted shapes are still on a voxel grid [23].

Parametric shapes offer a sparse, non-voxelized solution. Methods for converting point clouds to geometric primitives achieve high-quality results but require supervision, either relying on existing labeled data [27, 26, 15] or prescribed templates [14]. Groueix et al. output primitives at any resolution, but their primitives are not naturally parameterized or sparsely represented [17]. Genova et. al. propose to represent geometry as isosurfaces of axis-aligned Gaussians [16]. Others [17, 39, 31] develop tailored primitives but use standard Chamfer distance as the loss objective. We demonstrate and address the issues inherent in Chamfer distance.

**Font exploration and manipulation.** Designing or even finding a font can be tedious using generic vector graphics tools. Certain geometric features distinguish letters from one another across fonts, while others distinguish fonts from one another. Due to these difficulties and the presence of large font datasets, font exploration, design, and retrieval have emerged as challenging problems in graphics and learning.

Previous exploration methods categorize and organize fonts via crowdsourced attributes [28] or embed fonts on a manifold using purely geometric features [6, 2]. Instead, we leverage deep vectorization to automatically generate a sparse representation for each glyph. This enables exploration on the basis of general shape rather than fine detail.

Automatic font generation methods usually fall into two categories. Rule-based methods [40, 32] use engineered decomposition and reassembly of glyphs into parts. Deep learning approaches [1, 43] produce raster images, with limited resolution and potential for image-based artifacts, making them unfit for use as glyphs. We apply our method to edit existing fonts while retaining vector structure and demonstrate vectorization of glyphs from noisy partial data.

**Parametric shape collections.** As the number of publicly-available 3D models grows, methods for organizing, classifying, and exploring models become crucial. Many approaches decompose models into modular parametric components, commonly relying on prespecified templates or labeled collections of specific parts [20, 36, 29]. Such shape collections prove useful in domain-specific applications in design and manufacturing [34, 42]. Our deep learning pipeline allows generation of parametric shapes to perform these tasks. It works quickly on new inputs at test time and is generic, handling a variety of modalities without supervision and producing different output types.

3. Preliminaries

Let $A, B \subset \mathbb{R}^n$ be two measurable shapes. Let $X$ and $Y$ be two point sets sampled uniformly from $A$ and $B$. The *directed Chamfer distance* between $X$ and $Y$ is

$$
\text{Ch}_{\text{dir}}(X, Y) = \frac{1}{|X|} \sum_{x \in X} \min_{y \in Y} \|x - y\|_2^2,
$$

(1)

and the *symmetric Chamfer distance* is defined as

$$
\text{Ch}(X, Y) = \text{Ch}_{\text{dir}}(X, Y) + \text{Ch}_{\text{dir}}(Y, X). 
$$

(2)

These were proposed for computational applications in [5] and have been used as a loss function assessing similarity of a learned shape to ground truth in learning [41, 12, 24, 17].

To relate our proposed loss to Chamfer distance, we define *variational directed Chamfer distance* as

$$
\text{Ch}_{\text{dir}}^{\text{var}}(A, B) = \frac{1}{\text{Vol}(A)} \int_{A} \inf_{y \in B} \|x - y\|_2^2 \, dV(x),
$$

(3)

with *variational symmetric Chamfer distance* $\text{Ch}(A, B)^{\text{var}}$ defined analogously, extending (1) and (2) to smooth objects.
If points are sampled uniformly, under relatively weak assumptions, \( \text{Ch}(X, Y) \to 0 \) iff \( A = B \) as the number of samples grows, making it a reasonable shape matching metric. Chamfer distance, however, has fundamental drawbacks:

- It is highly dependent on the sampled points and sensitive to non-uniform sampling, as in Figure 1a.
- It is agnostic to normal alignment. As in Figure 1b, Chamfer distance between a dense set of vertical lines and a dense set of horizontal lines approaches zero.
- It is slow to compute. For each \( x \) sampled from \( A \), it is necessary to find the closest \( y \) sampled from \( B \), a quadratic-time operation when implemented naively. Efficient structures like k-d trees are not well-suited to GPUs.

Our method does not suffer from these disadvantages.

4. Method

We introduce a framework for formulating loss functions suitable for learning parametric shapes in 2D and 3D; our formulation not only generalizes Chamfer distance but also leads to stronger loss functions that improve performance on a variety of tasks. We start by defining a general loss on distance fields and propose two specific losses.

4.1. General Distance Field Loss

Given \( A, B \subseteq \mathbb{R}^n \), let \( d_A, d_B : \mathbb{R}^n \to \mathbb{R}_+ \) measure distance from each point in \( \mathbb{R}^n \) to \( A \) and \( B \), respectively, \( d_A(x) := \inf_{y \in A} ||x - y||_2 \). In our experiments, \( n \in \{2, 3\} \).

Let \( S \subseteq \mathbb{R}^n \) be a bounded set with \( A, B \subseteq S \). We define a general distance field loss as

\[
\mathcal{L}_\Psi[A, B] = \frac{1}{\text{Vol}(S)} \int_{x \in S} \Psi_{A, B}(x) \, dV(x),
\]

(4)

for some measure of discrepancy \( \Psi \). Note that we represent \( A \) and \( B \) only by their respective distance functions, and the loss is computed over \( S \).

Let \( \Phi \in \mathbb{R}^p \) be a collection of parameters defining a shape \( S_\Phi \subseteq \mathbb{R}^n \). For instance, if \( S_\Phi \) consists of Bézier curves, \( \Phi \) contains a list of control points. Given a target shape \( T \subseteq \mathbb{R}^n \), we formulate fitting a parametric shape to approximate \( T \) w.r.t. \( \Psi \) as minimizing

\[
f_\Psi(\Phi) = \mathcal{L}_\Psi[S_\Phi, T].
\]

(5)

For optimal shape parameters, \( \hat{\Phi} := \arg \min_\Phi f_\Psi(\Phi) \). We propose two discrepancy measures, providing loss functions that capture different geometric features.

4.2. Surface Loss

We define surface discrepancy to be

\[
\Psi_{A,B}^{\text{surf}}(x) = \delta \{ \ker d_A \} (x) d_B^2(x) + \delta \{ \ker d_B \} (x) d_A^2(x)
\]

(6)

where \( \delta \{ X \} \) is the Dirac delta defined uniformly on \( X \), and \( \ker f \) denotes the zero level-set of \( f \). \( \Psi_{A,B}^{\text{surf}} > 0 \) iff the shapes do not match, making it sensitive to local geometry.

4.3. Normal Alignment Loss

We define normal alignment discrepancy to be

\[
\Psi_{A,B}^{\text{align}}(x) = 1 - \langle \nabla d_A(x), \nabla d_B(x) \rangle^2.
\]

(7)

Minimizing \( f_{\Psi_{A,B}^{\text{align}}} \) aligns normals of the predicted primitives to those of the target. Following Figure 1b, if \( A \) contains dense vertical lines and \( B \) contains horizontal lines, \( \mathcal{L}_{\Psi_{A,B}^{\text{align}}} \gg 0 \) while \( \text{Ch}(A, B) \approx 0 \).

4.4. Final Loss Function

The general distance field loss and proposed discrepancy measures are differentiable w.r.t. the shape parameters \( \Phi \), as long as \( d_{S_\Phi} \) is differentiable w.r.t. \( \Phi \). Thus, they are well-suited to be optimized by a deep network predicting parametric shapes. We approximate (4) via Monte Carlo integration:

\[
\mathcal{L}_\Psi[A, B] \approx \frac{1}{|G|} \sum_{x \in G} \Psi_{A,B}(x),
\]

(8)

where \( G \) is a 2D or 3D grid.

While we use a voxel grid grid to compute the integrals in our loss function, the resolution of the voxel grid only affects quadrature without limiting the resolution of our representation. The grid dictates how we sample distance values; the values themselves are derived from a continuous parametric representation. A small subvoxel change in the geometry will affect the distance value at multiple discrete voxels. This property is in distinct contrast to representations that only consider the occupancy grid of a shape—the resolution of such representations is strictly limited by the grid resolution.

For \( \Psi_{A,B}^{\text{surf}} \), we use Smootherstep\((1 - d_A^2 / \gamma^2)\) (with Smootherstep defined as in [11]) as a smooth version of
\[ \delta \{ \ker d^2_A \} \] to evaluate the expression on a grid and to avoid discontinuities, enabling smooth gradients in our optimization. We set \( \gamma \) to twice the diameter of a voxel. For \( \Psi^{\text{align}} \), we approximate gradients using finite differences.

We minimize \( f_\Psi = f_{\text{out}} + \alpha^{\text{align}} f_{\text{align}} \), determining \( \alpha^{\text{align}} = 0.01 \) for all experiments using cross-validation.

4.5. Network Architecture and Training

The network takes a 128 \times 128 image or a 64 \times 64 \times 64 distance field as input and outputs a parametric shape. We encode our input to a \( \mathbb{R}^{256} \) latent space using a ResNet-18 [18] architecture. We then use a fully connected layer with 256 units and ReLU nonlinearity followed by a fully connected layer with number of units equal to the dimension of the target parameterization. We pass the output through a sigmoid and rescale it depending on the parameters being predicted. Our pipeline is illustrated in Figure 2. We train each network on a single Tesla GeForce GTX Titan X GPU for approximately one day, using Adam [21] with learning rate \( 10^{-4} \) and batch size 32 for 2D and 16 for 3D.

5. 2D: Font Exploration and Manipulation

We demonstrate our method in 2D for font glyph vectorization. Given a raster image of a glyph, our network outputs control points defining a collection of quadratic Bézier curves that approximate its outline. We produce nearly exact vector representations of glyphs from simple (non-decorative) fonts. From a decorative glyph with fine-grained detail, however, we recover a good approximation of the glyph’s shape using a small number of Bézier primitives and a consistent structure. This process can be interpreted as projection onto a common latent space of control points.

We first describe our choice of primitives as well as the computation of their distance fields. We introduce a template-based approach to allow our network to better handle multimodal data (different letters) and test several applications.

5.1. Approach

Primitives. We wish to use a 2D parametric shape primitive that is sparse and expressive and admits an analytic distance field. Our choice is the quadratic Bézier curve (which we refer to as curve), parameterized by control points \( a, b, c \in \mathbb{R}^2 \) and defined by \( \gamma(t) = (1-t)^2a + 2(1-t)t b + t^2 c, \) for \( 0 \leq t \leq 1 \). We represent 2D shapes as the union of \( n \) curves parameterized by \( \Phi = \{ a_i, b_i, c_i \}_{i=1}^n \subseteq \mathbb{R}^{3n} \).

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figure3}
\caption{Glyphs with predicted boundary curves rendered with predicted stroke thickness. The network thickens curves to account for stylistic details at the glyph boundaries.}
\end{figure}

Proposition 2 Given a curve \( \gamma \) parameterized by \( a, b, c \in \mathbb{R}^2 \) and a point \( p \in \mathbb{R}^2 \), the \( \hat{t} \in \mathbb{R} \) such that \( \gamma(\hat{t}) \) is the closest point on the curve to \( p \) satisfies the following:
\begin{align}
(B, B) \hat{t}^3 + 3(A, B) \hat{t}^2 + 2(A, A) + (B, a-p) \hat{t} + (A, a-p) &= 0, \tag{9} \\
\end{align}
where \( A = b - a \) and \( B = c - 2b + a \).

Thus, evaluating the distance to a single curve \( d_{\gamma_i}(p) = \| p - \gamma_i(\hat{t}) \|_2 \) requires finding the roots of a cubic [33], which we can do analytically. To compute distance to the union of the curves, we take a minimum: \( d_{\Phi}(p) = \min_{i=1}^n d_{\gamma_i}(p) \).

In addition to the control points, we predict a stroke thickness for each curve. We use this parameter when computing the loss by “lifting” the predicted distance field, thus thickening the curve—if curve \( \gamma \) has thickness \( s \), we set \( d_{\gamma}(p) = \min(d_{\gamma_i}(p) - s, 0) \). While we do not visualize stroke thickness in our experiments, this approach allows the network to thicken curves to better match high-frequency filigree (see Figure 3). This thickening is a simple operation in our distance field representation; sampling-based methods do not provide a natural way to thicken predicted geometry.

Templates. Our training procedure is self-supervised, as we do not have ground truth curve annotations. To better handle the multimodal nature of our entire dataset with a single network, we label each training example with its letter, passed as additional input. This allows us to condition on input class by concatenating a 26-dimensional one-hot vector to the input, a common technique for conditioning [44].

We choose a “standard” curve representation per letter, capturing each letter’s distinct geometric and topological features, by designing 26 templates from a shared set of control points. A template of type \( \ell \in \{ A, \ldots, Z \} \) is a collection of points \( T_{\ell} = \{ p_1, \ldots, p_n \} \subseteq \mathbb{R}^{2n} \) with corresponding connectivity determining how the points define curves. Since our curves form closed loops, we reuse endpoints.

For glyph boundaries of uppercase English letters, there are three connectivity types—one loop (e.g., “C”), two loops (e.g., “A”), and three loops (“B”). In our templates, the first loop has 15 curves and the other loops have 4 curves each. We will show that while letter templates (Figure 4a) better specialize to the boundaries of each glyph, we still achieve good results with simple templates (Figure 4b). Even without letter-specific templates, our system learns a consistent geometric representation, establishing cross-glyph correspondences purely using self-supervision.

We use predefined templates together with our labeling of each training example for two purposes. First, connectivity
is used to compute curve control points from the network output. Second, they provide a template loss:

$$L_{\text{template}}(\ell, x) = \alpha_{\text{template}} e^{t/s} \| T_\ell - h^t(x) \|_2^2,$$

(10)

where $s \in \mathbb{Z}_{+}$, $\gamma \in (0, 1)$, $t$ is the iteration number, $x$ is the input image, and $h^t(x)$ is the network output at iteration $t$. This initializes the network output, such that an input of type $\ell$ initially maps to template $\ell$. As this term decays, the other loss terms take over. We set $\alpha_{\text{template}} = 10$ and $s = 500$, though other choices of parameters for which the template term initially overpowers the rest of the loss also work.

5.2. Experiments

We train our network on the 26 uppercase English letters extracted from nearly 10,000 fonts. The input is a raster image of a letter, and the target distance field to the boundary of the original vector representation is precomputed.

Ablation study. We demonstrate the benefit of our loss over Chamfer distance as well as the contribution of each of our loss terms. While having 26 unique templates helps achieve better results, it is not crucial—we evaluate a network trained with three “simple templates” (Figure 4b), which capture the three topology classes of our data.

<table>
<thead>
<tr>
<th>Model</th>
<th>Average error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full model (ours)</td>
<td>0.509</td>
</tr>
<tr>
<td>No surface term (ours)</td>
<td>1.613</td>
</tr>
<tr>
<td>No alignment term (ours)</td>
<td>0.642</td>
</tr>
<tr>
<td>Simple templates (ours)</td>
<td>0.641</td>
</tr>
<tr>
<td>Chamfer (with letter templates)</td>
<td>0.623</td>
</tr>
<tr>
<td>AtlasNet [17]</td>
<td>5.154</td>
</tr>
</tbody>
</table>

Table 1: Comparison between subsets of our full model as well as standard Chamfer distance and AtlasNet. Average error is Chamfer distance (in pixels on a $128 \times 128$ image) between ground truth and uniformly sampled predicted curves.

For the Chamfer loss experiment, we use the same hyperparameters as for our method and sample 5,000 points from the source and target geometry. We initialize the model output to the full letter templates, like in our full model.

We also evaluate on 20 sans-serif fonts, computing Chamfer distance between our predicted curves and ground truth geometry, sampling uniformly (average error in Table 1). Uniform sampling is a computationally-expensive and non-differentiable procedure only for evaluation a posteriori—not suitable for training. While it does not correct all of Chamfer distance’s shortcomings, we use it as a baseline to evaluate quality. We limit to sans-serif fonts since we do not expect to faithfully recover local geometry. Our full loss outperforms Chamfer loss, and both our loss terms are necessary. Figure 5 shows qualitative results on test set glyphs; see supplementary material for additional results.

Comparison to AtlasNet. In AtlasNet [17], geometry is reconstructed by training implicit decoders, which map a point in the unit square to a point on the target surface, optimizing Chamfer distance. We modify the AtlasNet system to our task and demonstrate that our method proposes a more effective geometry representation and loss.

AtlasNet represents shapes as points in a learned high dimensional space, which does not obviously correlate to geometric features. Thus, in contrast to our explicit representation as a collection of control points, it does not facilitate geometric interpretability. Additionally, this makes it difficult to impose geometric priors—it is unclear how to initialize AtlasNet to predefined templates, as we do in §5.1.
Figure 7: Vectorization of various glyphs. For each we show the raster input (top left, black) along with the vectorization (colored curves) superimposed. When the input has simple structure (a), we recover an accurate vectorization. For fonts with decorative details (b), our method places curves to capture overall structure. Results are taken from the test dataset.

Figure 8: Comparison to AtlasNet [14] with a closed loop start shape to our simple templates and full models. We only train (and test) AtlasNet on letters with a single loop. For a fair comparison, we train an AtlasNet model that maps points from the boundary of a circle (rather than the interior of a square) into 2D. We only train on letters with single loop topology (C, E, F, etc.) and sample 5,000 points. Thus, this setting is comparable to the simple templates experiment from our ablation.

We show results in Figure 8. Although AtlasNet recovers overall structure of the input, it suffers from artifacts, self-intersections, and imprecision not exhibited by our method, even with simple templates. Likely, this is due to the fact that AtlasNet exhibits the drawbacks of Chamfer distance identified in §3, i.e., non-uniform sampling and lack of sensitivity to normal alignment. We include a quantitative comparison in Table 1. Our method outperforms AtlasNet even based on a uniformly-sampled Chamfer distance metric.

Vectorization. For any font glyph, our method generates a consistent sparse vector representation, robustly and accurately describing the glyph’s structure while ignoring decorative and noisy details. For simple fonts, our representation is a near-perfect vectorization, as in Figure 7a. For decorative glyphs, our method produces a meaningful abstraction. While a true vectorization would contain many curves with a large number of connected components, we succinctly capture the glyph’s overall structure (Figure 7b).

Figure 9: Nearest neighbors for a glyph in curve space, sorted by proximity. The query glyph is in orange.

Nearest-neighbor lookups in curve space also can help find a font matching desired geometric characteristics. A possible workflow is in Figure 11—through incremental refinements of the curves the user can quickly find a font.

Our method preserves semantic correspondences. The same curve is consistently used for the boundary of, e.g., the top of an “I”. These correspondences persist across letters with both full and simple templates—see, e.g., the “E” and “F” in Figure 7a and 7b and “simple templates” in Figure 5.

Retrieval and exploration. Our sparse representation can be used to explore the space of glyphs, useful for artists and designers, without the need for manual labelling or annotation. Treating control points as a metric space, we can perform Euclidean nearest-neighbor lookups for font retrieval.

In Figure 9, for each query glyph, we compute its curve representation and retrieve seven nearest neighbors in curve space. Because our representation captures geometric structure, we find glyphs that are similar structurally, despite decorative and stylistic differences.

We can also consider a path in curve space starting at the curves for one glyph and ending at those for another. By sampling nearest neighbors along this trajectory, we “interpolate” between glyphs. As in Figure 10, this produces meaningful collections of fonts for the same letter and reasonable results when the start and end glyphs are different letters. Additional results are in supplementary material.

Style and structure mixing. Our sparse curve representation describes geometric structure, ignoring stylistic and
Figure 10: Interpolating between fonts in curve space. The start and end are in orange and blue, respectively, and the nearest glyphs to linear interpolants are shown in order.

Figure 11: User-guided font exploration. At each edit, the nearest glyph is displayed below. This lets the user explore the dataset through geometric refinements.

Figure 12: Mixing of style (columns) and structure (rows) of the A glyph from different fonts. We deform each starting glyph (orange) into the structure of each target glyph (blue).

Figure 13: Vectorized GAN-generated fonts from [1].

Our system learns a strong prior on glyph shape, allowing us to robustly handle noisy input. In [1], a generative adversarial network (GAN) generates novel glyphs. The outputs, however, are raster images, often with noise and missing parts. Figure 13 shows how our method can simultaneously vectorize and repair GAN-generated glyphs. Compared to a vectorization tool like Adobe Illustrator Live Trace, we infer missing data based on learned priors, making the glyphs usable starting points for font design.

6.3D: Volumetric Primitive Prediction

We reconstruct 3D surfaces out of various primitives, which allow our model to be expressive, sparse, and abstract.

6.1. Approach

Our first primitive is a **cuboid**, parameterized by \( \{b, t, q\} \), where \( b = (w, h, d) \), \( t \in \mathbb{R}^3 \) and \( q \in \mathbb{S}^4 \) a quaternion, i.e., an origin-centered (hollow) rectangular prism with dimensions \( 2b \) to which we apply rotation \( q \) and then translation \( t \).

**Proposition 3** Let \( C \) be a cuboid with parameters \( \{b, t, q\} \) and \( p \in \mathbb{R}^3 \) a point. Then, the signed distance between \( p \) and \( C \) is

\[
d_C(p) = \|\max(d, 0)\|_2 + \min(\max(d_x, d_y, d_z), 0),
\]

where \( p' = q^{-1}(p - t)q \) using the Hamilton product and \( d = (|p_x'|, |p_y'|, |p_z'|) - b \).

Inspired by [31], we additionally use a **rounded cuboid** primitive by introducing a radius parameter \( r \) and computing the signed distance by \( d_{RC}(p) = d_C(p) - r \).

A unique advantage of our distance field representation is the ability to perform CSG boolean operations. Since our distances are signed, we can compute the distance to the union of \( n \) primitives by taking a minimum over distance...
fields. With sampling-based methods such as Chamfer distance optimization, care must be taken to avoid sampling interior faces that are not part of the outer surface.

6.2. Experiments

We train on the airplane and chair categories of ShapeNet Core V2 [7], taking as input a distance field. Thus, our method is fully self-supervised.

Surface abstraction. In Figure 15, for each ShapeNet chair, we show the our cuboid abstraction, our rounded cuboid abstraction, and the abstraction of [41]. We show our cuboid abstractions of ShapeNet airplanes in Figure 16. Each of our networks outputs 16 primitives, and we discard cuboids with high overlap using the method of [41]. The resulting abstractions capture high-level structures of the input. See supplementary material for additional results.

Segmentation. Because we place cuboids consistently, we can use them for segmentation. Following [41], we demonstrate on the COSEG chair dataset. We first label each cuboid predicted by our network (trained on ShapeNet chairs) with a segmentation class (seat, back, legs). Then, we generate a cuboid decomposition of each chair in the dataset and segment according to the nearest cuboid. We achieve a mean accuracy of 94.6%, exceeding the 89.0% accuracy of [41].

CSG operations. In Figure 17, we show results of a network that outputs parameters for the union of eight rounded cuboids minus eight other rounded cuboids. For inputs compatible with this template, we get good results. It is unclear how to achieve unsupervised CSG predictions using Chamfer loss.

7. Conclusion

Representation is a key theme in deep learning—and machine learning more broadly—applied to geometry. Assorted means of communicating a shape to and from a deep network present varying tradeoffs between efficiency, quality, and applicability. While considerable effort has been put into choosing representations for certain tasks, the tasks we consider have fixed representations for the input and output: They take in a shape as a function on a grid and output a sparse set of parameters. Using distance fields and derived functions as intermediate representations is natural and effective, not only performing well empirically but also providing a simple way to describe geometric loss functions.

Our learning procedure is applicable to many additional tasks. A natural next step is to incorporate our network into more complex pipelines for tasks like vectorization of complex drawings [3], for which the output of a learning procedure needs to be combined with classical techniques to ensure smooth, topologically valid output. A challenging direction might be to incorporate user guidance into training or evaluation, developing the algorithm as a partner in shape reconstruction rather than generating a deterministic output.

Our experiments suggest several extensions for future work. The key drawback of our approach is the requirement of closed-form distances for the primitives. While there are many primitives that could be incorporated this way, a fruitful direction might be to alleviate this requirement, e.g. by including flexible implicit primitives like metaballs [4]. We could also incorporate more boolean operations into our pipeline, which easily supports them using algebraic operations on signed distances, in analogy to the CAD pipeline, to generate complex topologies and geometries with few primitives. The combinatorial problem of determining the best sequence of boolean operations for a given input would be particularly challenging even for clean data [10]. Finally, it may be possible to incorporate our network into generative algorithms to create new unseen shapes.

8. Acknowledgement

The authors acknowledge the generous support of Army Research Office grant W911NF1710068, Air Force Office of Scientific Research award FA9550-19-1-031, of National Science Foundation grant IIS-1838071, National Science Foundation Graduate Research Fellowship under Grant No. 1122374, from an Amazon Research Award, from the MIT-IBM Watson AI Laboratory, from the Toyota-CSAIL Joint Research Center, from a gift from Adobe Systems, and from the Skoltech-MIT Next Generation Program.
References


