Deep Parametric Shape Predictions using Distance Fields

Dmitriy Smirnov[1], Matthew Fisher[2], Vladimir G. Kim[2], Richard Zhang[2], Justin Solomon[1]

[1]Massachusetts Institute of Technology, [2]Adobe Research

Abstract

Many tasks in graphics and vision demand machinery for converting shapes into 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 raster grid. We demonstrate efficacy on 2D and 3D tasks, including font vectorization and surface abstraction.

1. Introduction

The creation, modification, and rendering of vector graphics and parametric shapes is a fundamental problem of interest to engineers, artists, animators, and designers. Such representations offer distinct advantages over other models. By expressing a shape as a collection of primitives, we are able to apply transformations easily, to identify correspondences, and to render at arbitrary resolution, all while having to store only a sparse representation.

It is often useful to generate a parametric model from data that does not directly correspond to the target geometry and contains 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 [23], segmentation [27], and image-to-image translation [20]. CNNs, however, operate on *raster* representations. Grid structure is fundamentally built into convolution as a mechanism for information to travel between layers of a deep network. This structure is leveraged during training to optimize performance on a GPU. Recent deep learning pipelines that output vector shape primitives [40] have been significantly less successful than pipelines for analogous tasks on raster images or voxelized volumes.

A challenge when applying deep learning to parametric geometry is the combination of Eulerian and Lagrangian representations. CNNs process data in an *Eulerian* fashion in that they apply fixed operations to a dense grid of values; Eulerian shape representations like indicator functions come as values on a fixed grid. Parametric shapes, on the other hand, use sparse sets of parameters like control points to express geometry. In contrast to stationary Eulerian grid points, this *Lagrangian* representation moves with the shape. Navigating the transition from Eulerian to Lagrangian geometry is a key step in any learning pipeline for the problems above, a task we consider in detail.

We propose a deep learning framework for predicting parametric shapes, addressing the aforementioned issues. By analytically computing a distance field to the primitives at each training iteration, we formulate an Eulerian version of the Chamfer distance, a common metric for geometric similarity. Our metric can be computed efficiently and does not require sampling points from the predicted or target shapes. Beyond accelerating evaluation of existing loss functions, our distance field enables alternative loss functions that are sensitive to specific geometric qualities like alignment.

We apply our new framework in the 2D context to a diverse dataset of fonts. We train a network that takes in a raster image of a glyph and outputs a representation as a collection of Bézier curves. This maps glyphs onto a common set of parameters that can be traversed intuitively. We use this embedding for font exploration and retrieval, correspondence, and unsupervised interpolation.

We also show that our approach works in 3D. With surface primitives in place of curves, we perform volumetric abstraction on ShapeNet [8], inputting an image or a distance field and outputting parametric primitives that approximate

the model. This output can be rendered at any resolution or converted to a mesh; it also can be used for segmentation.

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

- a *general distance field loss function* allowing definition of several 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

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 [29] or embed fonts on a manifold using purely geometric features [7, 3]. 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 [39, 32] use engineered decomposition and reassembly of glyphs into parts. Deep learning approaches [1, 42] 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 partial and noisy data, *i.e.*, raster images from a generative model.

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 [21, 36, 30]. Such shape collections prove useful in domain-specific applications in design and manufacturing [34, 41]. 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.

Deep 3D reconstruction. Combining multiple viewpoints to reconstruct 3D geometry is crucial in applications like robotics and autonomous driving [15, 35, 38]. Even more challenging is inference of 3D structure from one input. Recent deep networks can produce point clouds or voxel occupancy grids given a single image [14, 10], but their output suffers from fixed resolution.

Learning signed distance fields defined on a voxel grid [12, 37] or directly [31] 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 [25]

Parametric 3D 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 data labelled with primitives [24, 28] or prescribed templates [16]. Tulsiani *et al*. output cuboids but are limited in output type [40]. Groueix *et al*. output primitives at any resolution, but their primitives are not naturally parameterized or sparsely represented [17].

3. Preliminaries

Let $A, B \subset \mathbb{R}^n$ be two smooth (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

$$\mathrm{Ch}_{\mathrm{dir}}(X, Y) = \sum_{x \in X} \min_{y \in Y} \mathrm{d}(x, y), \tag{1}$$

and the *symmetric Chamfer distance* is defined as

$$\mathrm{Ch}(X, Y) = \mathrm{Ch}_{\mathrm{dir}}(X, Y) + \mathrm{Ch}_{\mathrm{dir}}(Y, X). \tag{2}$$

It was proposed for computational applications in [6] and has been used as a loss function assessing similarity of a learned shape to ground truth in deep learning [40, 14, 26, 17].

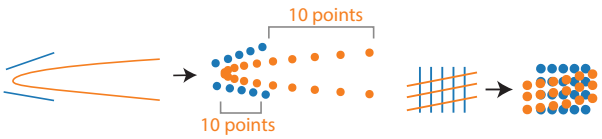
We also define *variational directed Chamfer distance*

$$\mathrm{Ch}^{\mathrm{var}}_{\mathrm{dir}}(A, B) = \int_A \inf_{y \in B} \|x - y\|_2 \, \mathrm{d}x, \tag{3}$$

with *variational symmetric Chamfer distance* $\mathrm{Ch}(A, B)^{\mathrm{var}}$ defined analogously, extending (1) and (2) to smooth objects. We use this to relate our proposed loss to Chamfer distance.

If points are sampled uniformly, under relatively weak assumptions about A and B, $\mathrm{Ch}(X, Y) \to 0$ iff $A = B$, as the sizes of the sampled point sets grow. Thus, it is 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 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 naïvely. Efficient structures like k-d trees are not well-suited to GPUs.



(a) Sampling (b) Alignment

Figure 1: Drawbacks of Chamfer distance. In (a), sampling from Bèzier curve B (blue) by uniformly sampling in parameter space yields disproportionately many points at the high-curvature area, resulting in a low Chamfer distance to the segments of A (orange) despite geometric dissimilarity. In (b), two sets of nearly-orthogonal line segments have near-zero Chamfer distance despite misaligned normals.

- It is agnostic to normal alignment. As in Figure 1b, the Chamfer distance between a dense set of vertical lines and a dense set of horizontal lines approaches zero.

Our method does not suffer from these disadvantages.

4. Method

Beyond architectures for particular tasks, we introduce a framework for formulating loss functions suitable for learning placement of parametric shapes in 2D and 3D; our formulation not only encapsulates Chamfer distance—and suggests a means of accelerating its computation—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 three specific losses.

4.1. General Distance Field Loss

Given $A, B \subseteq \mathbb{R}^n$, let $\mathrm{d}_A, \mathrm{d}_B : \mathbb{R}^n \to \mathbb{R}_+$ measure distance from each point in $\mathbb{R}^n$ to A or B, respectively, $\mathrm{d}_A(x) := \inf_{y \in A} \|x - y\|_2$. In our experiments, $n \in \{2, 3\}$. Let $S \subseteq \mathbb{R}^n$ be a bounded set with $A, B \subseteq S$. We define the *general distance field loss* as

$$\mathcal{L}_\Psi[A, B] = \int_{x \in S} \Psi_{A,B}(x)\, \mathrm{d}V(x), \tag{4}$$

for some measure of discrepancy Ψ. 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. For instance, a parametric shape may consist of Bézier curves, in which case Φ contains a list of control points. Let $\mathrm{d}_\Phi : \mathbb{R}^n \to \mathbb{R}_+$ be the distance to the shape defined by Φ. Given a target object T with distance function d_T, we formulate fitting a parametric shape to approximate T w.r.t. Ψ as minimizing

$$f_\Psi(\Phi) = \mathcal{L}_\Psi[\Phi, T]. \tag{5}$$

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

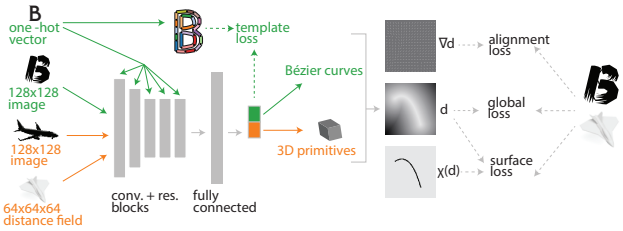


Figure 2: An overview of our pipelines—font vectorization (green) and volumetric primitive prediction (orange).

4.2. Surface Loss

We define surface discrepancy to be

$$\Psi^{\text{surf}}_{A,B}(x) = \delta\{\ker \mathrm{d}_A\}(x)\mathrm{d}_B(x) + \delta\{\ker \mathrm{d}_B\}(x)\mathrm{d}_A(x) \tag{6}$$

where $\delta\{X\}$ is the Dirac delta defined uniformly on X, and $\ker f$ denotes the zero level-set of f. Ψ^{surf} is only nonzero where the shapes do not match, making it sensitive to fine details in the matching:

Proposition 1 *The symmetric variational Chamfer distance between $A, B \subset \mathbb{R}^n$ equals the corresponding surface loss between,* i.e., $\mathrm{Ch}^{\text{var}}(A, B) = \mathcal{L}_{\Psi^{surf}_{A,B}}$.

Unlike the Chamfer distance, the discrete version of our surface loss can be approximated efficiently on GPU without sampling points from either the parametric or target shape.

4.3. Global Loss

We define global discrepancy to be

$$\Psi^{\text{glob}}_{A,B}(x) = |\,\mathrm{d}_A(x) - \mathrm{d}_B(x)|^2. \tag{7}$$

Minimizing $\mathcal{L}_{\Psi^{\text{glob}}_{A,B}}$ is equivalent to minimizing the L_2 distance between d_A and d_B. $f_{\Psi^{\text{glob}}}(\Phi)$ increases quadratically in distance between the shape defined by Φ and the target shape. Thus, minimizing $f_{\Psi^{\text{glob}}}$ encourages global alignment, quickly placing the parametric primitives close to the target and accelerating convergence.

4.4. Normal Alignment Loss

We define normal alignment discrepancy to be

$$\Psi^{\text{align}}_{A,B}(x) = \|\nabla\, \mathrm{d}^2_A(x) - \nabla\, \mathrm{d}^2_B(x)\|^2_2. \tag{8}$$

Minimizing $f_{\Psi^{\text{align}}}$ 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, $\mathcal{L}_{\Psi^{\text{align}}_{A,B}}$ is large while $\mathrm{Ch}(A, B) \approx 0$.

4.5. Final Loss Function

The general distance field loss and the specific discrepancy measures proposed thus far are differentiable w.r.t the

shape parameters Φ, as long as the distance function d_Φ is differentible w.r.t. Φ. Thus, they are well-suited to be optimized by a deep network that predicts parametric shapes. To discretize, we simply redefine (4) to be

$$\mathcal{L}_\Psi[A, B] = \sum_{x \in G} \Psi_{A,B}(x), \tag{9}$$

where G is a 2D or 3D grid. Thus, we minimize a weighted sum of $f_\Psi(\Phi)$ across the Ψ defined in §4:

$$\Psi = \Psi^{\text{glob}} + \alpha^{\text{surf}}\Psi^{\text{surf}} + \alpha^{\text{align}}\Psi^{\text{align}}. \tag{10}$$

We use $\alpha^{\text{surf}} = 1$ and $\alpha^{\text{align}} = 0.001$ across experiments. In 3D experiments, we decay the global loss term exponentially by a factor 0.5 every 500 iterations.

4.6. Network architecture

The network takes a 128×128 image or a $64 \times 64 \times 64$ distance field as input and outputs a parametric shape. We use the same architecture for 2D and 3D, following advances in network architecture design. Let `c5s2-64` be a convolutional layer with 64 filters of 5×5 evaluated at stride 2, `Rx7` be 7 residual blocks [19, 18] of size 3×3 (keeping filter count constant), and `fc-512` be a fully-connected layer. To increase receptive field without dramatically increasing parameter count, we also use dilated convolution [43, 9] in residual blocks. We use `ELU` [11] after all linear layers except the last. We use `LayerNorm` [2] after each conv and residual layer, except the first. Our encoder architecture is: `c5s1-32`, `c3s2-64`, `c3s1-64`, `c3s2-128`, `c3s1-128`, `c3s1-128`, `c3s2-256`, `Rx7`, `c3s2-256`, `Rx1`, `c3s2-256`, `Rx1`, `fc-512`, `fc-N`, where `N` is the dimension of our target parameterization. Our pipeline is illustrated in Figure 2. We train each network on a single Tesla K80 GPU, using Adam [22] with learning rate 10^{-4} and batch size 16.

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 that form a collection of quadratic Bézier curves approximating its outline. When used on a glyph of a simple font (non-decorative, *e.g.*, sans-serif), our method recovers nearly the exact original vector representation. From a decorative glyph with fine-grained detail, however, we recover a good approximation of the glyph's shape using relatively few Bézier primitives and a consistent structure. This process can be interpreted as projection onto a common sparse 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 multi-modal data (different letters) and test several applications.

5.1. Approach

5.1.1 Primitives

We wish to use a 2D parametric shape primitive that is sparse and expressive and admits an analytic distance field. Our choice satisfying these requirements is the *quadratic Bèzier curve*, which we will refer to as *curve*, parameterized by control points $a, b, c \in \mathbb{R}^2$ and defined by $B(t) = (1-t)^2 a + 2(1-t)tb + t^2 c$, for $0 \le t \le 1$. We represent 2D shapes as the union of n curves parameterized by $\Phi = \{a_1, b_1, c_1, \ldots, a_n, b_n, c_n\}$, where $a_i, c_i, b_i \in \mathbb{R}^2$.

Proposition 2 *Given a curve B 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 $B(\hat{t})$ is the closest point on the curve to p satisfies the following:*

$$\begin{aligned}\langle B, B\rangle \hat{t}^3 + 3\langle A, B\rangle \hat{t}^2 + (2\langle A, A\rangle + \langle B, a-p\rangle)\hat{t} \\ + \langle A, a-p\rangle = 0,\end{aligned} \tag{11}$$

where $A = b - a$ and $B = c - 2b + a$.

Thus, evaluating the distance to a single curve $\mathrm{d}(p, B_i) = \|p - B_i(\hat{t})\|_2$ requires finding the roots of a cubic [33], which we can do analytically in constant time. To compute distance to the union of the curves, we take a minimum:

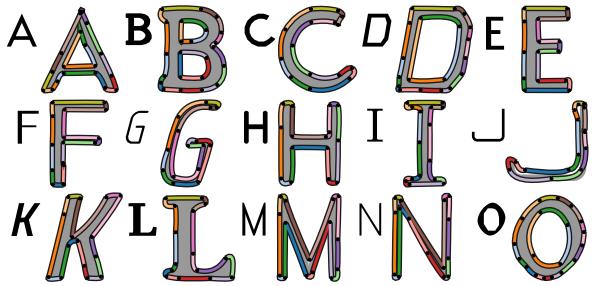
$$\mathrm{d}_\Phi(p) = \min_{i=1}^{n} \mathrm{d}_{B_i}(p). \tag{12}$$

In addition to the control points, we predict a stroke thickness parameter for each curve. We use this parameter when computing the loss by "lifting" the predicted distance field and, consequently, thickening the curve—if a curve B has stroke thickness s, we set $d_B^s(p) = \min(d_B(p) - s, 0)$. While we do not visualize stroke thickness in our experiments, this approach allows the network to thicken curves to better match fine-grained decorative details (Figure 4). This thickening is a simple and natural operation in our distance field representation; note that sampling-based methods do not provide a natural way to "thicken" the surfaces.

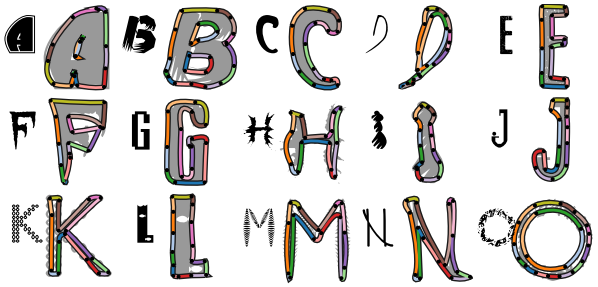
5.1.2 Templates

Our training procedure is unsupervised, as we do not have ground truth curve annotations. To better handle the multi-modal nature of our data without a separate network for each letter, we label each training example with its letter, passed as additional input to our network. This allows us to condition based on input class by concatenating a 26-dimensional one-hot vector to the input of each convolutional layer (after replicating to the appropriate spatial dimensions), a common technique for conditioning [44].

We choose a "standard" Bèzier curve representation for each letter, which captures that letter's distinctive geometric and topological features, by designing 26 templates from



(a) Plain font glyphs



(b) Decorative font glyphs

Figure 3: Vectorization of various glyphs. For each we show the raster input (top left, gray) 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 template curves to capture overall structure. Results are taken from the test dataset.

Figure 4: Glyphs with corresponding predicted curves rendered with predicted stroke thickness. The network thickens curves of to account for stylistic details.

(a) Letter templates

(b) Simple templates

Figure 5: Font glyph templates. These determine the connectivity and initialize the placement of the predicted curves.

a shared set of control points output by our network. A *template* of class $\ell \in \{\mathrm{A}, \ldots, \mathrm{Z}\}$ is a collection of points $T_\ell = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^{2n}$ with corresponding *connectivity* determining how points in T_ℓ are used to define curves. Since we predict glyph boundaries, our final curves form closed loops, allowing us to reuse endpoints.

For extracting glyph boundaries from uppercase English letters, there are three connectivity types—one loop (*e.g.*, *C*), two loops (*e.g.*, *A*), and three loops (*B*). We design templates such that the first loop has 15 curves and the other loops have 4 curves each. Our templates are shown in Figure 5. We will show that while letter templates (a) are better able to specialize to the boundaries of each glyph, we still achieve good results for most letters with the simple templates (b), which also allow for establishing cross-glyph correspondences.

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*

$$\mathcal{L}^{\text{template}}(c, x) = \alpha^{\text{template}} \gamma^{(t/s)} \|T_c - h^t(x)\|_2^2, \qquad (13)$$

where $s \in \mathbb{Z}_+$, $\gamma \in (0, 1)$, and t is the current iteration. This serves to initialize the network output, such that a training example of class ℓ initially maps to template letter ℓ; as the loss decays during training, it acts as a regularizer. We choose $\alpha^{\text{template}} = 1$, $\gamma = 0.7$, and $s = 300$.

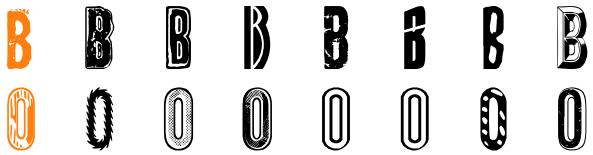


Figure 6: Nearest neighbors for a glyph in curve space, sorted by proximity. The query glyph is in orange.

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.

5.2.1 Vectorization

For any font glyph, our method generates a sparse vector representation, which robustly and accurately describes the glyph's structure while ignoring decorative and noisy details. For simple fonts comprised of few strokes, our representation is a nearly perfect vectorization, as in Figure 3a.

For glyphs from decorative fonts, our method produces a meaningful representation. In such cases, a true vectorization would contain many curves with a large number of connected components. Our network places the sparse curve template to best capture the glyph's structure, as in Figure 3b.

Our method preserves semantic correspondences in our templates. The same curve is consistently used for the boundary of, *e.g.*, the top of an *I*. These correspondences persist *across* letters for both letter templates and simple templates—see for example the *E* and *F* glyphs in Figure 3a and 3b and "simple templates" in Figure 12.

We demonstrate robustness in Figure 8 by quantizing our loss values and visualizing the number of examples for each value. Outliers corresponding to higher losses are generally caused by noisy data—they are either not uppercase English letters or have fundamentally uncommon structure.

Figure 7: Interpolating between fonts in curve space. The start and end are shown in orange and blue, resp., and nearest-neighbor glyphs to linear interpolants are shown in order.

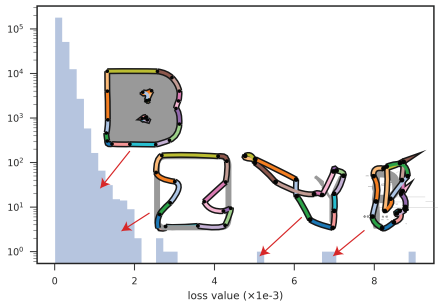


Figure 8: Number of examples per quantized loss value. We visualize the input and predicted curves for several outliers.

5.2.2 Retrieval and Exploration

Our sparse representation can be used to explore the space of glyphs, useful for artists and designers. By treating the control points as a metric space, we can perform nearest-neighbor lookups to retrieve fonts using Euclidean distance.

In Figure 6, for each query, we compute its curve representation and retrieve seven nearest neighbors in curve space. Because our representation uses geometric structure, we find fonts 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 7, 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 available in the supplementary material.

Nearest-neighbor lookups in curve space also can help find a font matching desired geometric characteristics. A possible workflow is in Figure 9—through incremental refinements of the curves the user can quickly find a font.

5.2.3 Style and Structure Mixing

Our sparse curve representation describes geometric structure, ignoring stylistic elements (*e.g.*, texture, decorative details). We leverage this to warp a glyph with desired style to have a target structure of another glyph (Figure 10).

We first generate the sparse curve representation for source and target glyphs. Since our representation uses the same set of curves, we can estimate dense correspondences and use them to warp original vectors of source glyph to conform to the shape of the target. For each point p on the source, we find the closest point q on the its sparse curve representation. We then compute the translation from q to the corresponding point on the target glyph's sparse representation and apply this translation to p.

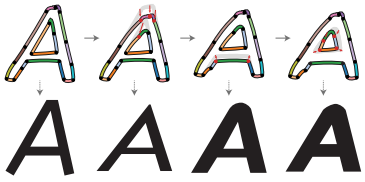


Figure 9: User-guided font exploration. At each edit, the nearest-neighbor glyph is displayed on the bottom. This lets the user explore the dataset through geometric refinements.

Figure 10: 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).

5.2.4 Repair

Our system introduces a strong prior on glyph shape, allowing us to robustly handle noisy input. In [1], a generative adversarial network (GAN) generates new glyphs based on samples. The outputs, however, are raster images, often with noise and missing parts. Figure 11 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 to reasonably fit the template. This post-processing step makes the glyphs from generative models usable starting points for font design.

5.2.5 Comparison and Ablation Study

We show a series of experiments that demonstrate the advantages of our loss over standard Chamfer distance as well as

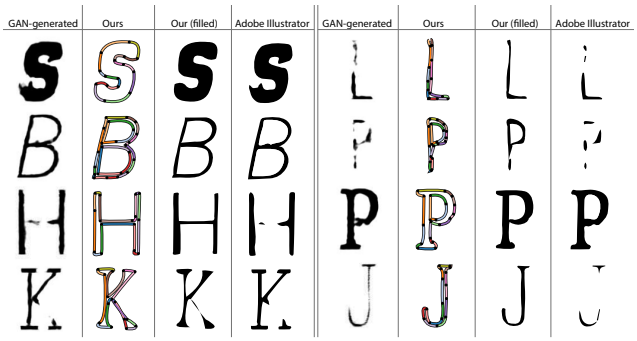


Figure 11: Vectorization of GAN-generated fonts from [1].

Loss type	Sec/it	Average error
Full model (ours)	1.119	0.748
No global term (ours)	1.111	0.817
No surface term (ours)	1.109	0.761
No alignment term (ours)	1.118	0.786
Simple templates (ours)	1.127	0.789
Chamfer	1.950	0.910

Table 1: Comparison between subsets of our full loss as well as standard Chamfer distance. Average error is Chamfer distance (in pixels on a 128×128 image) between predicted curves and ground truth, with points sampled uniformly. This demonstrates advantages of our loss over Chamfer distance and shows how each loss term contributes to the results.

the contributions of each term in our loss. We demonstrate that while having 26 unique templates helps achieve better results, it is not crucial—we evaluate a network trained with three "simple templates" (Figure 5b), which capture the three topology classes of our data.

Table 1 shows seconds per iteration for our full distance field loss, our loss without each of its three terms and with simple templates, and Chamfer loss. Training using our loss is nearly twice as fast as Chamfer per iteration. Moreover, each of our loss terms adds ≤ 0.1 seconds. In these experiments, for training with the distance field function we use the same parameters as above, and for Chamfer loss, we use the same training procedure (hyperparameters, templates, batch size), sampling 5,000 points from the source and target geometry. We choose the highest possible value for which the sampled point pairwise distance matrix fits in GPU memory.

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 the 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. We see our full loss outperforms Chamfer loss, and all three loss terms are

Figure 12: Comparisons to missing terms and Chamfer.

necessary. Here, all models are trained for 35,000 iterations. Figure 12 shows qualitative results on test set glyphs; see supplementary material for additional results.

The global loss term strongly favors rough similarity between predicted and targeted geometry, helping training converge more quickly (see supplementary material Figure 18).

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

Proposition 3 *Let C be a cuboid with parameters $\{b, t, r\}$ and $p \in \mathbb{R}^3$ a point. Let $p' = r^{-1}(p-t)r$ using the Hamilton product. Then, the signed distance between p and C is*

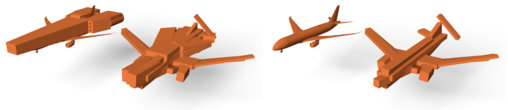
$$\mathrm{d}(p, C) = \|\max(d, 0)\|_2 + \min(\max(d_x, d_y, d_z), 0), \quad (14)$$

where $d = (|p'_x|, |p'_y|, |p'_z|) - b$ and v_x, v_y, v_z denote the x-, y-, and z-coordinates, respectively, of vector v.

Our second primitive is a sphere, parameterized by its center $c \in \mathbb{R}^3$ and radius $r \in \mathbb{R}$. We compute signed distance to the sphere S via the expression $\mathrm{d}(p, S) = \|p - c\|_2 - r$.

Since these distances are *signed*, we can compute the distance to the union of n primitives as in the 2D case, by taking a minimum over the individual primitive distances. Similarly, we can perform other boolean operations, *e.g.*, difference. The result is, again, a signed distance function, and so we take its absolute value before computing losses.

(a) ShapeNet chairs



(b) ShapeNet airplanes

(c) Shape COSEG chairs

Figure 13: Cuboid shape abstractions on test set inputs. In (a) and (b), we show ShapeNet chairs and airplanes. In (c), we show Shape COSEG chair segmentation. We show each input model (left) next to the cuboid representation (right).

6.2. Experiments

We train on the airplane and chair categories of ShapeNet [8] using distance fields from [12]. The input is a distance field. Hence, our method is self-supervised: The distance field is the only information needed to compute the loss. Additional results are available in supplementary material.

Surface abstraction. Figure 13 shows results of training shape-specific networks to build abstractions over chairs and airplanes from $64 \times 64 \times 64$ distance fields. Each network outputs 16 cuboids. We discard small cuboids with high overlap as in [40]. The resulting abstractions capture high-level structures of the input.

Segmentation. Because we place cuboids in a consistent way, we can use them for segmentation. Following [40], we demonstrate on the COSEG chair dataset. We first label each cuboid predicted by our network (trained on ShapeNet chairs) with one of the three segmentation classes in COSEG (seat, back, legs). Then, we generate a cuboid decomposition of each chair mesh in the dataset and segment by labelling each face according to its nearest cuboid (Figure 13c). We achieve a mean accuracy of 94.6%, exceeding the 89.0% accuracy reported by [40].

Single view reconstruction. In Figure 14, we show results of a network that takes as input a ShapeNet model rendering and outputs parameters for the union of four cuboids minus the union of four spheres. We see that for inputs that align well with this template, the network produces good results.

Figure 14: Single view reconstruction using different primitives and boolean operations.

It is not straightforward to achieve unsupervised CSG-style predictions using sampling-based Chamfer distance 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 through different discrepancies Ψ.

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 rasterization of complex drawings [4], 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 prediction or 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 [5]. 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 [13]. Finally, it may be possible to incorporate our network into *generative* algorithms to create new unseen shapes.

References

[1] S. Azadi, M. Fisher, V. Kim, Z. Wang, E. Shechtman, and T. Darrell. Multi-content gan for few-shot font style transfer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 11, page 13, 2018. 2, 6, 7

[2] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 4

[3] E. Balashova, A. Bermano, V. G. Kim, S. DiVerdi, A. Hertzmann, and T. Funkhouser. Learning a stroke-based representation for fonts. *CGF*, 2018. 2

[4] M. Bessmeltsev and J. Solomon. Vectorization of line drawings via polyvector fields. *ACM Transactions on Graphics (TOG)*, 2019. 8

[5] J. F. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics (TOG)*, 1(3):235–256, 1982. 8

[6] G. Borgefors. Distance transformations in arbitrary dimensions. *Computer vision, graphics, and image processing*, 27(3):321–345, 1984. 2

[7] N. D. Campbell and J. Kautz. Learning a manifold of fonts. *ACM Transactions on Graphics (TOG)*, 33(4):91, 2014. 2

[8] A. X. Chang, T. Funkhouser, L. 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. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015. 1, 8

[9] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2018. 4

[10] C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *European conference on computer vision*, pages 628–644. Springer, 2016. 2

[11] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015. 4

[12] A. Dai, C. R. Qi, and M. Nießner. Shape completion using 3d-encoder-predictor cnns and shape synthesis. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, volume 3, 2017. 2, 8

[13] T. Du, J. P. Inala, Y. Pu, A. Spielberg, A. Schulz, D. Rus, A. Solar-Lezama, and W. Matusik. Inversecsg: Automatic conversion of 3d models to csg trees. In *SIGGRAPH Asia 2018 Technical Papers*, page 213. ACM, 2018. 9

[14] H. Fan, H. Su, and L. J. Guibas. A point set generation network for 3d object reconstruction from a single image. In *CVPR*, volume 2, page 6, 2017. 2

[15] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha. Visual simultaneous localization and mapping: a survey. *Artificial Intelligence Review*, 43(1):55–81, 2015. 2

[16] V. Ganapathi-Subramanian, O. Diamanti, S. Pirk, C. Tang, M. Niessner, and L. Guibas. Parsing geometry using structure-aware shape templates. In *2018 International Conference on 3D Vision (3DV)*, pages 672–681. IEEE, 2018. 2

[17] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry. Atlasnet: A papier-mâché approach to learning 3d surface generation. *arXiv preprint arXiv:1802.05384*, 2018. 2

[18] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 4

[19] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016. 4

[20] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. *arXiv preprint*, 2017. 1

[21] 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. *ACM Transactions on Graphics (TOG)*, 32(4):70, 2013. 2

[22] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 4

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 1

[24] L. Li, M. Sung, A. Dubrovina, L. Yi, and L. Guibas. Supervised fitting of geometric primittives to 3d point clouds. *arXiv preprint arXiv:1811.08988*, 2018. 2

[25] Y. Liao, S. Donné, and A. Geiger. Deep marching cubes: Learning explicit surface representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2916–2925, 2018. 2

[26] X. Liu and K. Fujimura. Hand gesture recognition using depth data. In *Proc. 6th IEEE Int. Conf. Automatic Face Gesture Recog.*, page 529. IEEE, 2004. 2

[27] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015. 1

[28] C. Niu, J. Li, and K. Xu. Im2struct: Recovering 3d shape structure from a single rgb image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4521–4529, 2018. 2

[29] P. O'Donovan, J. Lībeks, A. Agarwala, and A. Hertzmann. Exploratory font selection using crowdsourced attributes. *ACM Transactions on Graphics (TOG)*, 33(4):92, 2014. 2

[30] M. Ovsjanikov, W. Li, L. Guibas, and N. J. Mitra. Exploration of continuous variability in collections of 3d shapes. In *ACM Transactions on Graphics (TOG)*, volume 30, page 33. ACM, 2011. 2

[31] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. *arXiv preprint arXiv:1901.05103*, 2019. 2

[32] H. Q. Phan, H. Fu, and A. B. Chan. Flexyfont: Learning transferring rules for flexible typeface synthesis. In *Computer Graphics Forum*, volume 34, pages 245–256. Wiley Online Library, 2015. 2

[33] Z. Qin, M. D. McCool, and C. S. Kaplan. Real-time texture-mapped vector glyphs. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 125–132. ACM, 2006. 4

[34] A. Schulz, A. Shamir, I. Baran, D. I. Levin, P. Sitthi-Amorn, and W. Matusik. Retrieval on parametric shape collections. *ACM Transactions on Graphics (TOG)*, 36(1):11, 2017. 2

[35] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *CVPR*, 2006. 2

[36] C.-H. Shen, H. Fu, K. Chen, and S.-M. Hu. Structure recovery by part assembly. *ACM Transactions on Graphics (TOG)*, 31(6):180, 2012. 2

[37] D. Stutz and A. Geiger. Learning 3d shape completion under weak supervision. *International Journal of Computer Vision*, pages 1–20, 2018. 2

[38] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 945–953, 2015. 2

[39] R. Suveeranont and T. Igarashi. Example-based automatic font generation. In *International Symposium on Smart Graphics*, pages 127–138. Springer, 2010. 2

[40] S. Tulsiani, H. Su, L. J. Guibas, A. A. Efros, and J. Malik. Learning shape abstractions by assembling volumetric primitives. In *Proc. CVPR*, volume 2, 2017. 1, 2, 8

[41] N. Umetani, T. Igarashi, and N. J. Mitra. Guided exploration of physically valid shapes for furniture design. *ACM Trans. Graph.*, 31(4):86–1, 2012. 2

[42] P. Upchurch, N. Snavely, and K. Bala. From a to z: supervised transfer of style and content using deep neural network generators. *arXiv preprint arXiv:1603.02003*, 2016. 2

[43] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015. 4

[44] J.-Y. Zhu, R. Zhang, D. Pathak, T. Darrell, A. A. Efros, O. Wang, and E. Shechtman. Toward multimodal image-to-image translation. In *Advances in Neural Information Processing Systems*, pages 465–476, 2017. 4

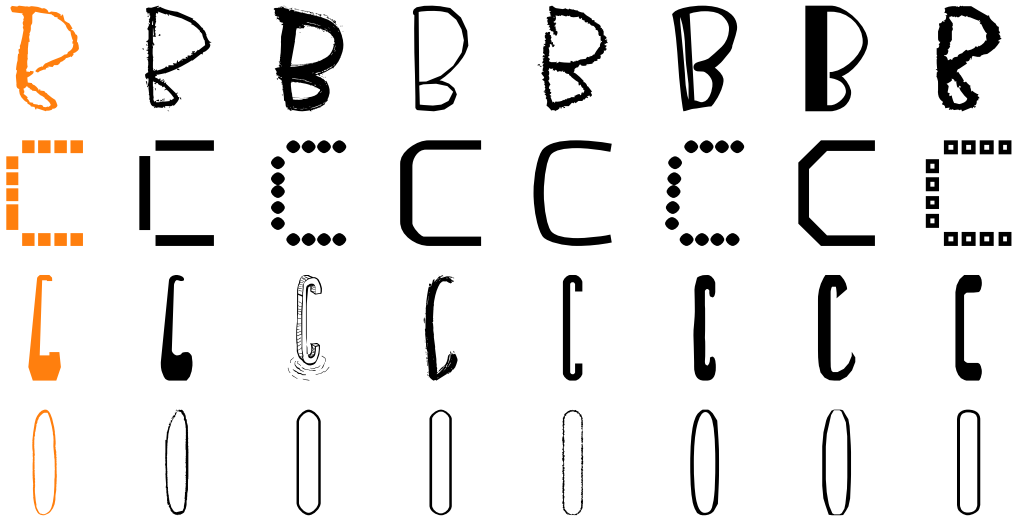


Figure 15: Glyph nearest neighbors in curve space.

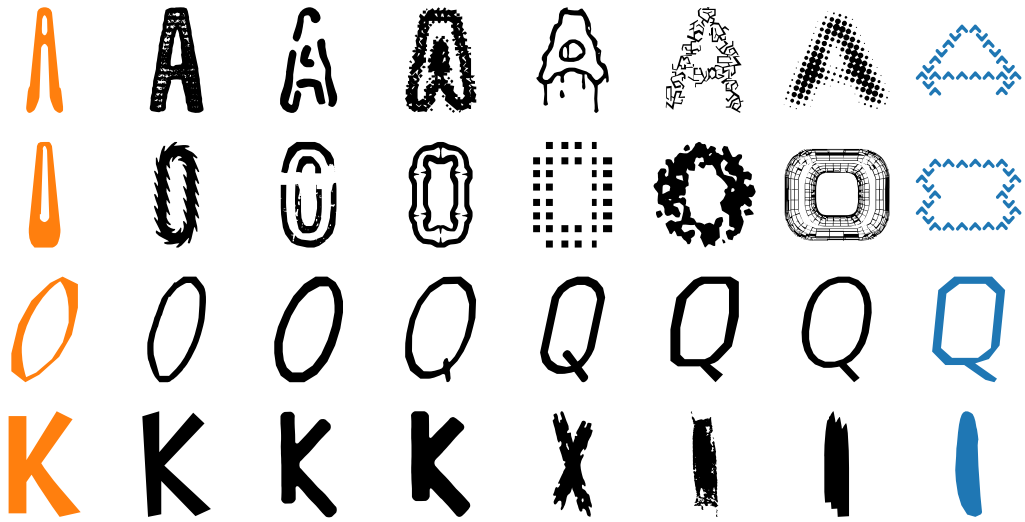


Figure 16: Interpolations between fonts in curve space.

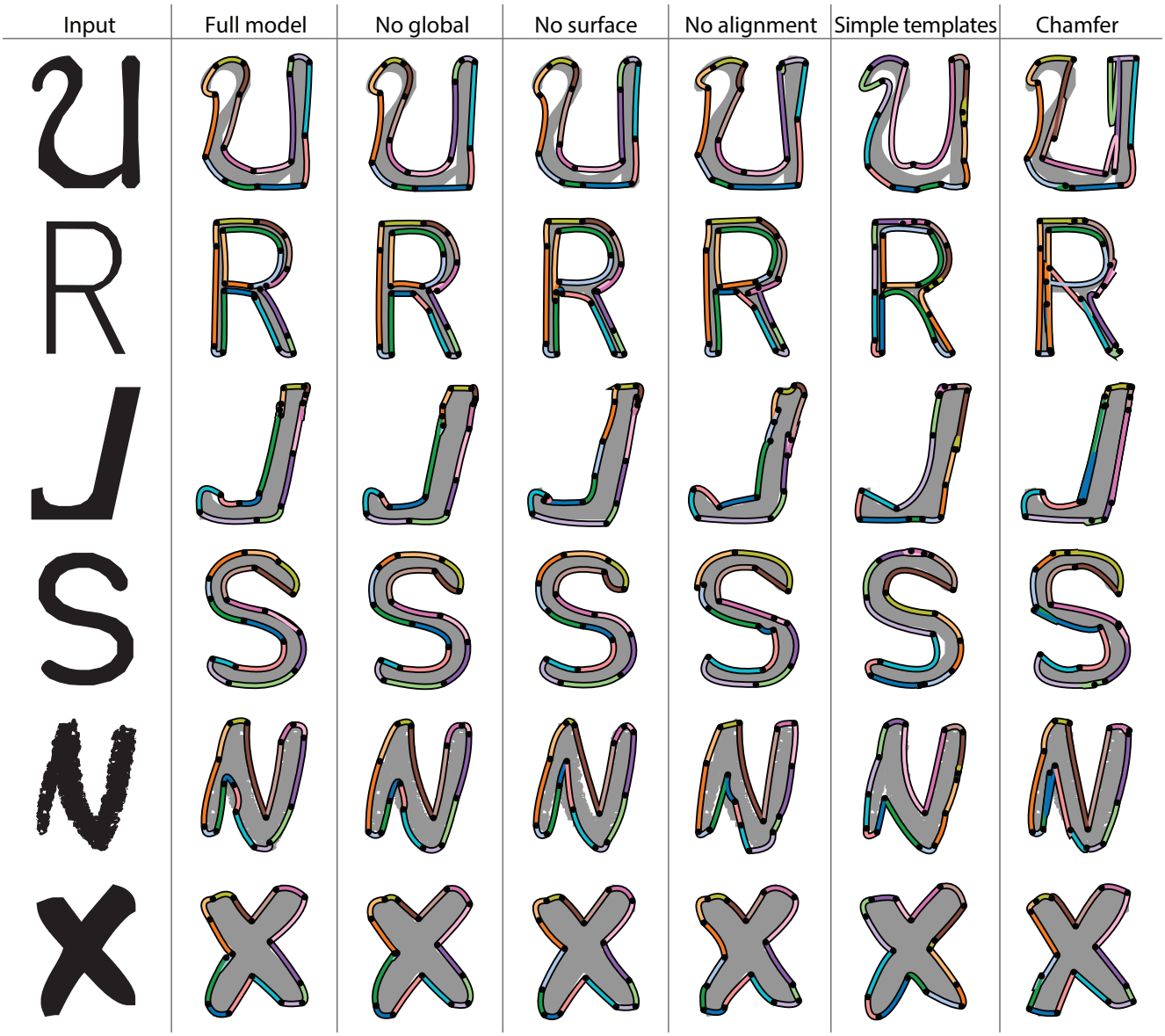


Figure 17: Distance field loss comparisons.

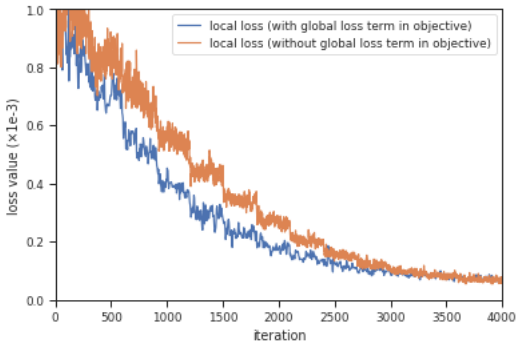


Figure 18: Local loss (smoothed) over the first 4,000 iterations of training with and without global loss in the objective function. The global loss term results in faster convergence.

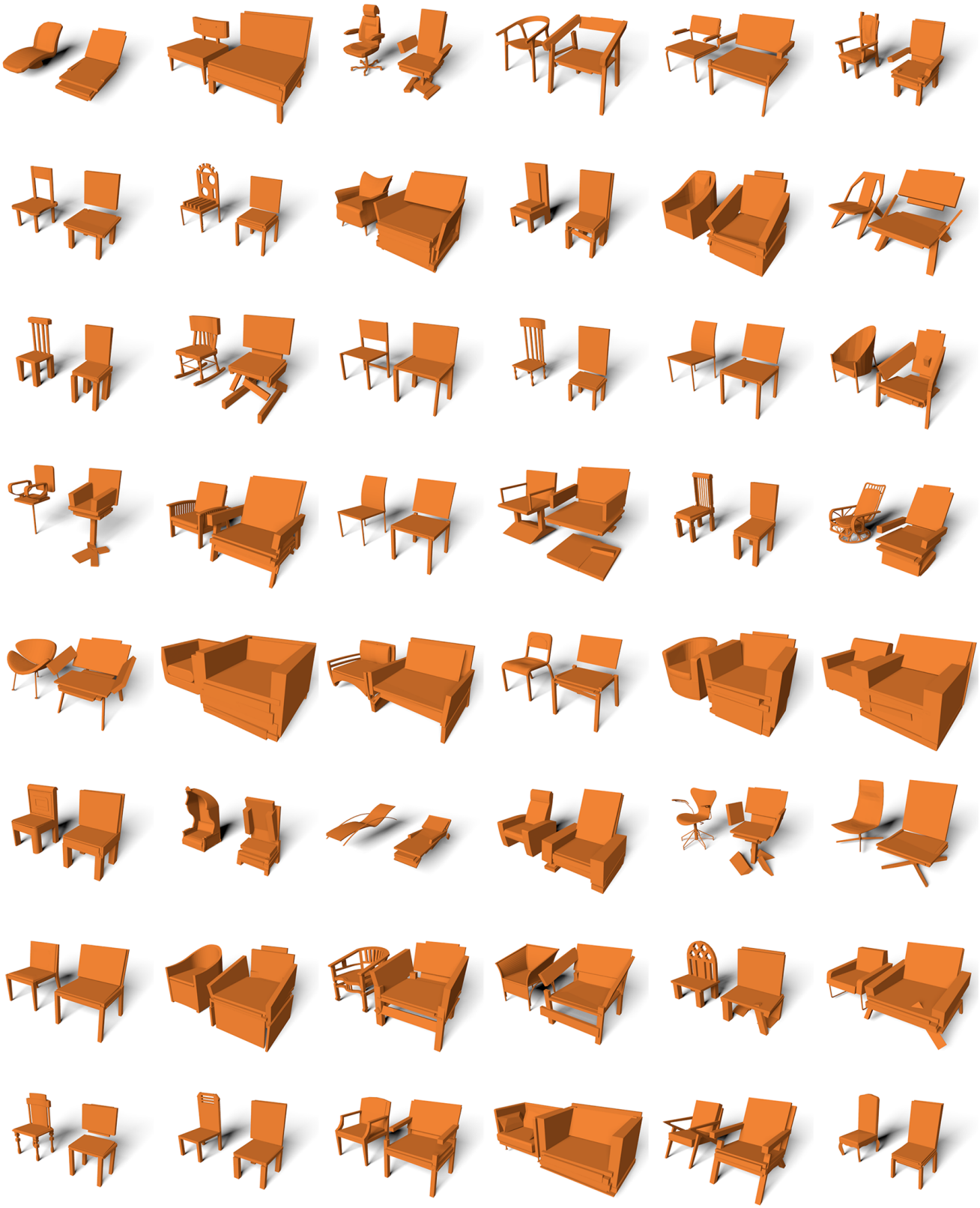


Figure 19: Cuboid reconstructions of ShapeNet chairs.

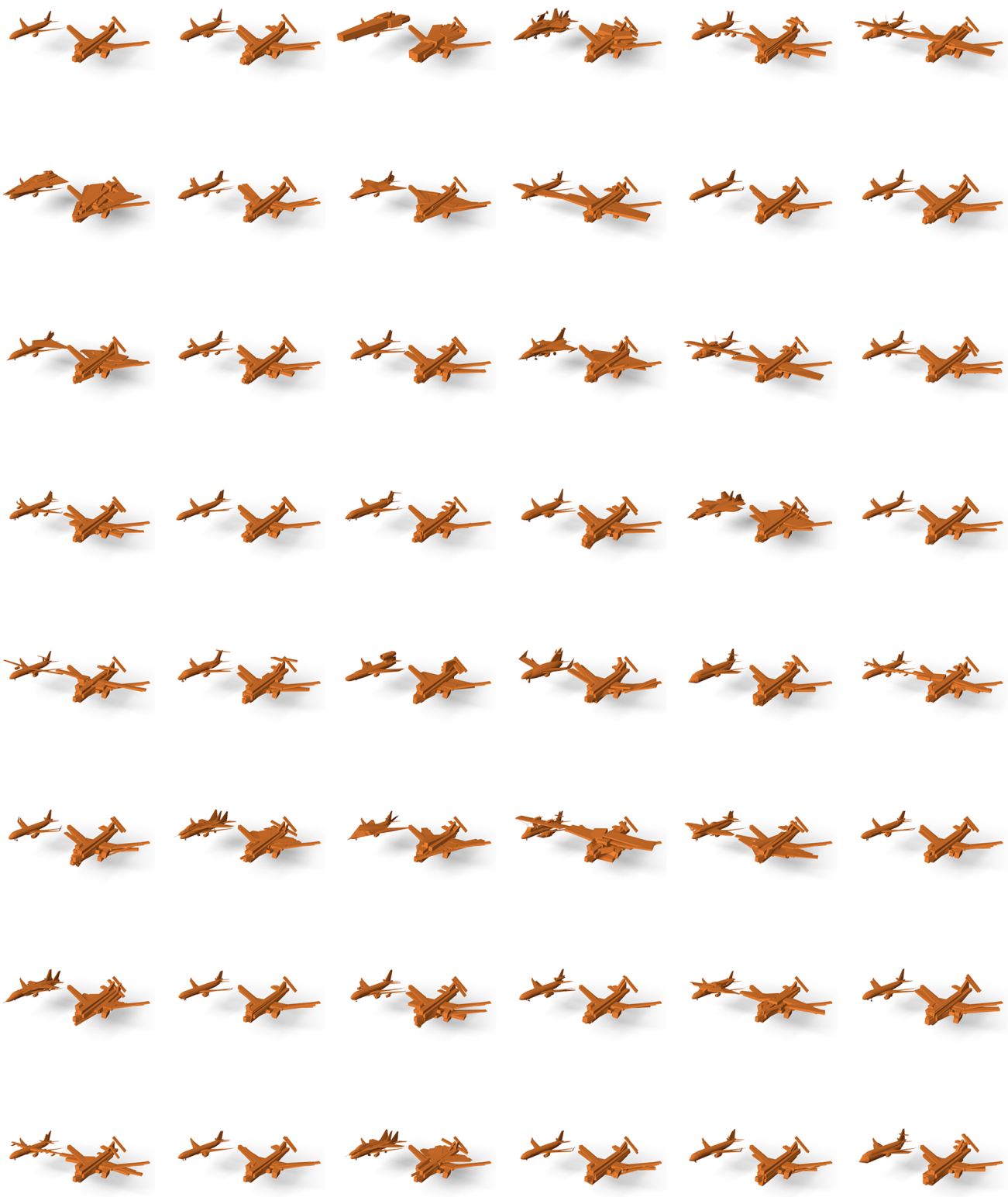


Figure 20: Cuboid reconstructions of ShapeNet airplanes.

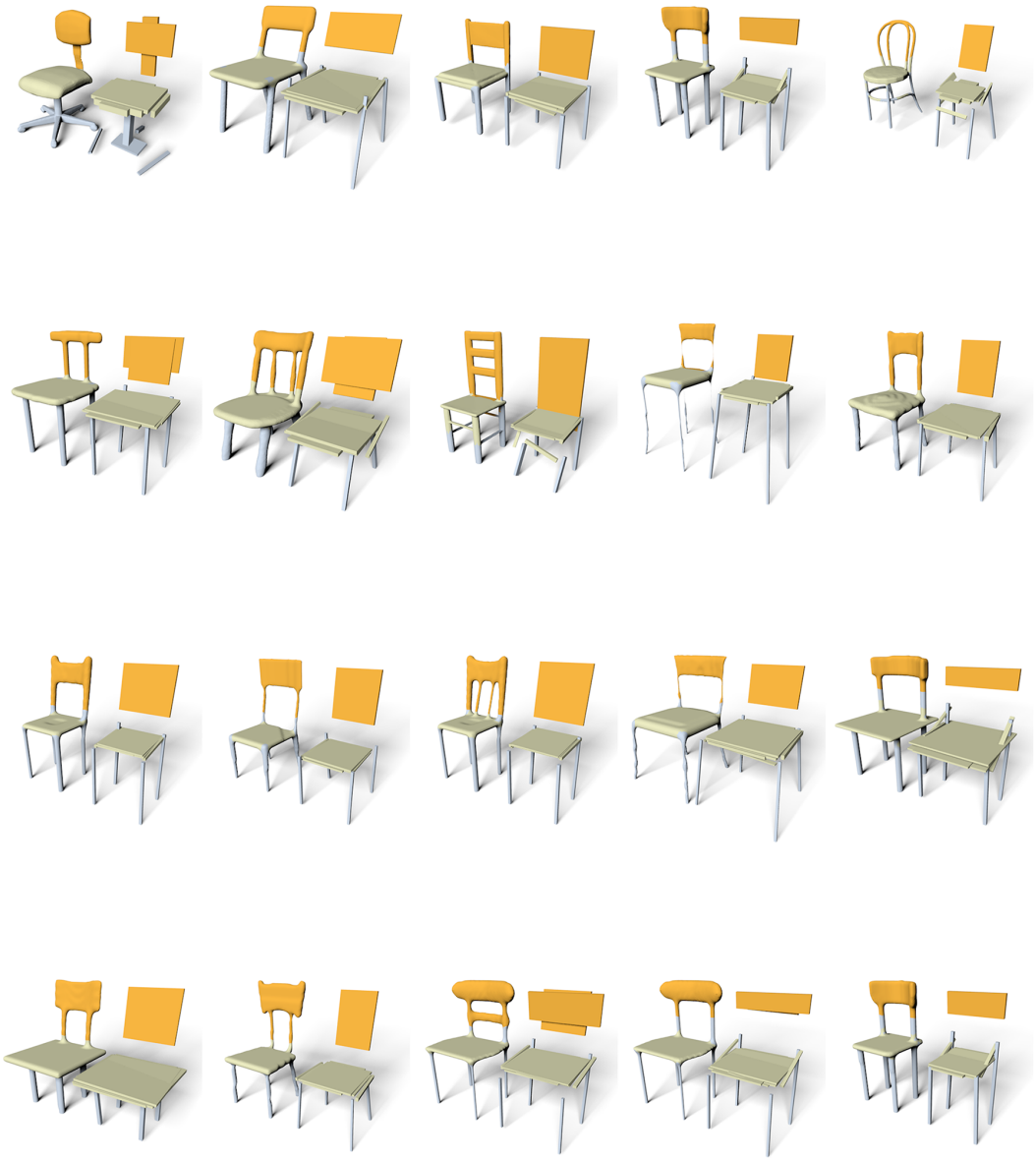


Figure 21: Cuboid segmentation of Shape COSEG chairs.