LEARNING MANIFOLD PATCH-BASED REPRESENTATIONS OF MAN-MADE SHAPES

Anonymous authors
Paper under double-blind review

ABSTRACT

Choosing the right representation for geometry is crucial for making 3D models compatible with existing applications. Focusing on piecewise-smooth man-made shapes, we propose a new representation that is usable in conventional CAD modeling pipelines and can also be learned by deep neural networks. We demonstrate its benefits by applying it to the task of sketch-based modeling. Given a raster image, our system infers a set of parametric surfaces that realize the input in 3D. To capture piecewise smooth geometry, we learn a special shape representation: a deformable parametric template composed of Coons patches. Naïvely training such a system, however, is hampered by non-manifold artifacts in the parametric shapes and by a lack of data. To address this, we introduce loss functions that bias the network to output non-self-intersecting shapes and implement them as part of a fully self-supervised system, automatically generating both shape templates and synthetic training data. We develop a testbed for sketch-based modeling, demonstrate shape interpolation, and provide comparison to related work.

1 INTRODUCTION

While state-of-the art deep learning systems that output 3D geometry as point clouds, triangle meshes, voxel grids, and implicit surfaces yield detailed results, these representations are dense, high-dimensional, and incompatible with CAD modeling pipelines. In this work, we develop a 3D representation that is parsimonious, geometrically interpretable, and easily editable with standard tools, while being compatible with deep learning. This enables a shape modeling system leveraging the ability of neural networks to process incomplete, ambiguous input and produces useful, consistent 3D output.

Our primary technical contributions involve the development of machinery for learning *parametric* 3D surfaces in a fashion that is efficiently compatible with modern deep learning pipelines and effective for a challenging 3D modeling task. We automatically infer a template per shape category and incorporate loss functions that operate explicitly on the geometry rather than in the parametric domain or on a sampling of surrounding space. Extending learning methodologies from images and point sets to more exotic modalities like networks of surface patches is a central theme of modern graphics, vision, and learning research, and we anticipate broad application of these developments in CAD workflows.

To test our system, we choose sketch-based modeling as a target application. Converting rough, incomplete 2D input into a clean, complete 3D shape is extremely ill-posed, requiring hallucination of missing parts and interpretation of noisy signal. To cope with these ambiguities, existing systems either rely on hand-designed priors, severely limiting applications, or learn the shapes from data, implicitly inferring relevant priors (Delanoy et al., 2018; Wang et al., 2018a; Lun et al., 2017). However, the output of the latter methods often lacks resolution and sharp features necessary for high-quality 3D modeling.

In industrial design, man-made shapes are typically modeled as collections of smooth parametric patches (e.g., NURBS surfaces) whose boundaries form the sharp features. To learn such shapes effectively, we use a deformable parametric template (Jain et al., 1998)—a manifold surface composed of patches, each parameterized by control points (Fig. 3a). This representation enables the model to control the smoothness of each patch and introduce sharp edges between patches where necessary.

Compared to traditional representations, deformable parametric templates have numerous benefits for our task. They are intuitive to edit with conventional software, are resolution-independent, and can be

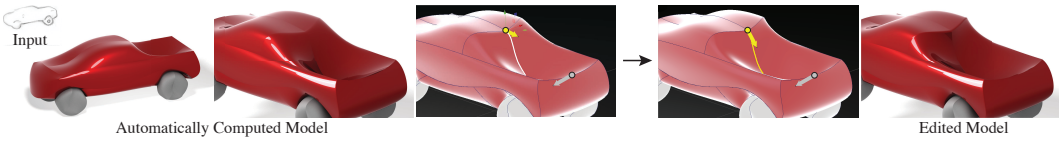


Figure 1: Editing a 3D model produced by our method. Because we output 3D geometry as a collection of consistent, well-placed NURBS patches, user edits can be made in conventional CAD software by simply moving control points. Here, we are able to refine the trunk of a car model with just a few clicks.

meshed to arbitrary accuracy. Since only boundary control points are needed, our representation has relatively few parameters. Finally, this structure admits closed-form expressions for normals and other geometric features, which can be used for loss functions that improve reconstruction quality (§3.2).

Training a model for such representations faces three major challenges: detection of non-manifold surfaces, structural variation within shape categories, and lack of data. We address them as follows:

- We introduce several loss functions that encourage our patch-based output to form a manifold mesh without topological artifacts or self-intersections.
- Some categories of man-made shapes exhibit structural variation. To address this, for each category we algorithmically generate a varying deformable template, which allows us to support separate structural variation using a variable number of parts (§3.1).
- Supervised methods mapping from sketches to 3D require a database of sketch-model pairs, and, to-date, there are no such large-scale repositories. We use a synthetic sketch augmentation pipeline inspired by artistic literature to simulate variations observed in natural drawings (§4.1). Although our model is trained on synthetic sketches, it generalizes to natural sketches.

Our method is self-supervised: We predict patch parameters, but our data is not labeled with patches.

2 Related Work

2.1 Deep learning for shape reconstruction

Learning to reconstruct 3D geometry has recently enjoyed significant research interest. Typical forms of input are images (Gao et al., 2019; Wu et al., 2017; Delanoy et al., 2018; Häne et al., 2019) and point clouds (Williams et al., 2019; Groueix et al., 2018; Park et al., 2019). When designing a network for this task, two considerations affect the architecture: the loss function and the geometric representation.

Loss Functions. One popular direction employs a differentiable renderer and measures 2D image loss between a rendering of the inferred 3D model and the input image (Kato et al., 2018; Wu et al., 2016; Yan et al., 2016; Rezende et al., 2016; Wu et al., 2017; Tulsiani et al., 2017b; 2018). A notable example is the work by Wu et al. (2017), which learns a mapping from a photograph to a normal map, a depth map, a silhouette, and the mapping from these outputs to a voxelization. They use a differentiable renderer and measure inconsistencies in 2D. Hand-drawn sketches, however, cannot be interpreted as perfect projections of 3D objects: They are imprecise and often inconsistent (Bessmeltsev et al., 2016). Another approach uses 3D loss functions, measuring discrepancies between the predicted and target 3D shapes directly, often via Chamfer or a regularized Wasserstein distance (Williams et al., 2019; Liu et al., 2010; Mandikal et al., 2018; Groueix et al., 2018; Park et al., 2019; Gao et al., 2019; Häne et al., 2019). We build on this work, adapting Chamfer distance to patch-based geometric representations and extending the loss function with new regularizers (§3.2).

Shape representation. As noted by Park et al. (2019), geometric representations in deep learning broadly can be divided into three classes: voxel-, point-, and mesh-based. Voxel-based methods (Delanoy et al., 2018; Wu et al., 2017; Zhang et al., 2018; Wu et al., 2018) yield dense reconstruction that are limited in resolution, offer no topological guarantees, and cannot represent sharp features. Point-based approaches represent geometry as a point cloud (Yin et al., 2018; Mandikal et al., 2018; Fan et al., 2017; Lun et al., 2017; Yang et al., 2018), sidestepping memory issues, but do not capture manifold connectivity.

Some recent methods represent shapes using meshes (Bagautdinov et al., 2018; Baque et al., 2018; Litany et al., 2018; Kanazawa et al., 2018; Wang et al., 2019; Nash et al., 2020). Our parametric template representation allows us to more easily enforce piecewise smoothness and test for self-

intersections (§3.2). These properties are difficult to measure on meshes in a differentiable manner. Compared to a generic template shape, such as a sphere, our *category-specific* templates improve reconstruction quality and enable complex reconstruction constraints, e.g., symmetry. We compare to the deformable mesh representations in §4.5.

Most importantly, our shape representation is *native* to modern CAD software and can be directly edited in this software, as demonstrated in Figure 1. The key to this flexibility is the type of the parametric patches we use, which can be trivially converted to a NURBS representation (Piegl & Tiller, 1996), the standard surface type in CAD. Other common shape representations, such as meshes or point clouds, cannot be easily converted into NURBS format: algorithmically fitting NURBS surfaces is nontrivial and is an active area of research (Yumer & Kara, 2012; Krishnamurthy & Levoy, 1996).

Finally, some recent works parameterize 3D geometry using learned deep neural networks—e.g., they learn an implicit representation (Mescheder et al., 2019; Chen & Zhang, 2019; Genova et al., 2019) or a mapping from parameter space to a collection patches (Groueix et al., 2018; Deng et al., 2020). These demonstrate impressive results, but are not tuned to CAD applications; it is unclear how their output can be converted to editable CAD shape representations.

2.2 SKETCH-BASED 3D SHAPE MODELING

3D reconstruction from sketches has a long history in graphics. A survey is beyond the scope of this paper; see (Delanoy et al., 2018) or surveys by Ding & Liu (2016) and Olsen et al. (2009).

Unlike incremental sketch-based 3D modeling, where users progressively add new strokes (Cherlin et al., 2005; Gingold et al., 2009; Chen et al., 2013; Igarashi et al., 1999), our method interprets complete sketches, eliminating training for artists and enabling 3D reconstruction of legacy sketches.

Some systems interpret complete sketches without extra information. This input is extremely ambiguous thanks to occlusions and inaccuracies. Hence reconstruction algorithms rely on strong 3D shape priors. These priors are typically manually created, e.g., for humanoids, animals, and natural shapes (Bessmeltsev et al., 2015; Entem et al., 2015; Igarashi et al., 1999). Our work focuses on man-made shapes, which have characteristic sharp edges and are only piecewise smooth. Rather than relying on expert-designed priors, we *automatically* learn category-specific shape priors.

A few deep learning approaches address sketch-based modeling. Nishida et al. (2016) and Huang et al. (2017) train networks to predict *procedural model parameters* that yield detailed shapes from a sketch. These methods produce complex high-resolution models, but only for shapes that can be procedurally generated. Lun et al. (2017) use a CNN-based architecture to predict multi-view depth and normal maps, later converted to point clouds; Li et al. (2018) improve on their results by first predicting a flow field from an annotated sketch. In contrast, we output a deformable parametric template, which can be converted to a manifold mesh without post-processing. Wang et al. (2018a) learn from unlabeled databases of sketches and 3D models with no correspondence using an adverserial training approach. Another inspiration for our research is the work of Delanoy et al. (2018), which reconstructs a 3D object as voxel grids; we compare to this work in Fig. 7.

3 ALGORITHM

Our learning pipeline outputs a parametrically-defined 3D surface. We describe our geometric representation (§3.1), define our loss (§3.2), and specify our architecture and training procedure (§3.3).

3.1 REPRESENTATION

Patches. Our surfaces are collections of *Coons patches* (Coons, 1967), a commonly used and rich subset of NURBS surfaces. A patch is specified by four boundary cubic Bézier curves sharing endpoints (Fig. 3a). Each curve has control points $p_1,\ldots,p_4 \in \mathbb{R}^3$. A Bézier curve $c:[0,1]\to\mathbb{R}^3$ is $c(\gamma)=p_1(1-\gamma)^3+3p_2\gamma(1-\gamma)^2+3p_3\gamma^2(1-\gamma)+p_4\gamma^3$, and a Coons patch $P:[0,1]^2\to\mathbb{R}^3$ is

$$P(s,t)=(1-t)c_1(s)+tc_3(1-s)+sc_2(t)+(1-s)c_4(1-t) \\ -(c_1(0)(1-s)(1-t)+c_1(1)s(1-t)+c_3(1)(1-s)t+c_3(0))\ st. \quad (1)$$

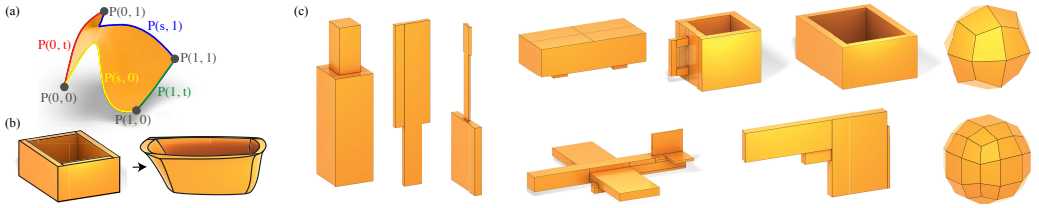
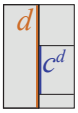


Figure 3: Our representation is composed of Coons patches (a) organized into a deformable template (b). We use the following templates (c, top to bottom, left to right): bottle, knife, guitar, car, airplane, coffee mug, gun, bathtub, 24-patch sphere, 54-patch sphere.

Templates. Templates specify patch connectivity. A template consists of the minimal number of control points necessary to define the patches; shared control points are reused (Fig. 3b, c). We allow the edge of one patch to be contained *within* the edge of another using *junction curves*. A *junction curve* c^d is constrained to a lie along a parent curve d and is thus parameterized by $s,t \in [0,1]$, such that $c(0) = d(s)$ and $c(1) = d(t)$.



A template provides hard topological constraints for our surfaces, an initialization of their geometry, and, optionally, a means for regularization. Templates are crucial in ensuring that the patches have consistent topology—an approach without templates would result in unstructured, non-manifold patch collections. While our method works using a generic sphere template, we can define templates per shape category to incorporate category-specific priors. These templates capture only coarse geometric features and approximate scale. We outline an algorithm for obtaining templates below.

Algorithmic construction of templates. We design a simple system to construct a template automatically given as input any collection of cuboids. Such a collection can be computed automatically for a shape category, e.g., given a segmentation or using self-supervised methods such as (Smirnov et al., 2020; Tulsiani et al., 2017a; Sun et al., 2019), or easily produced using standard CAD software. We show templates algorithmically computed from pre-segmented shapes—we obtain a collection of cuboids by taking the bounding box around each connected component of each segmentation class.

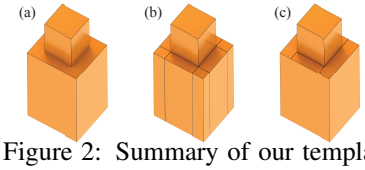


Figure 2: Summary of our template algorithm. Given a collection of cuboids (a), we form a quad mesh (b), and merge faces to get a template (c).

A generic cuboid decomposition cannot be used as a template, since individual cuboids may overlap. We snap cuboids to an integer lattice, split each face at grid coordinates, and remove overlapping and interior faces to obtain a manifold quad mesh. This mesh typically consists of many faces, and so, we simplify it. We merge adjacent quads with a greedy agglomerative algorithm, iterating over each quad in order of descending area and merging with an adjacent quad as long as the merge does not result in ill-defined junction curves. We show an example of this process in Fig. 2. Given decompositions of multiple shapes in a category, we use the median model w.r.t. Chamfer distance. Since models within a category are aligned, the median provides a rough approximation of the typical geometry.

Structural variation using templates. For category-specific templates, we use the fact that template patches are consistently placed on semantically meaningful parts to account for structural variation. For instance, certain airplane models contain turbines while others do not. We note which templates patches come from cuboids corresponding to turbines, and, during training, only use turbine patches for models that contain turbines. This allows to train the entire airplane shape category, effectively using two distinct templates. At test time, the user can toggle turbines on or off for any output model.

3.2 Loss

We fit a collection of patches $\{P_i\}$ to a target mesh M by optimizing a differentiable loss function. Below, we describe each term—a reconstruction loss generalizing Chamfer distance, a normal alignment loss, a self-intersection regularizer, a patch flatness regularizer, and two template-based priors.

Area-weighted Chamfer distance. Given two measurable shapes $A,B \subset \mathbb{R}^3$ and point sets X and Y sampled from A and B, respectively, the *directed Chamfer distance* between X and Y is

$$\mathrm{Ch}_{\text{dir}}(X,Y) = \frac{1}{|X|}\sum_{x\in X}\min_{y\in Y}\mathrm{d}(x,y), \tag{2}$$

where $\mathrm{d}(x,y) = \|x-y\|_2$. The *symmetric Chamfer distance* is $\mathrm{Ch}(X,Y) = \mathrm{Ch}_{\text{dir}}(X,Y) + \mathrm{Ch}_{\text{dir}}(Y,X)$.

Chamfer distance is differentiable and thus a popular loss function in deep learning. However, the distribution under which X and Y are sampled has significant impact. Sampling uniformly in patch parameter space (a unit square) does not capture the surface area measure and, consequently, does not yield uniform samples on the surface—regions with higher curvature end up oversampled. A closed form arc-length parameterization does not even exist for a Bézier curve, and so a uniform parameterization for even a one-dimensional patch is difficult to compute. Many works try to address this issue by designing custom sampling procedures (Wang et al., 2012; Hernández-Mederos & Estrada-Sarlabous, 2003; Elkott et al., 2002). These schemes are typically computationally expensive and/or incompatible with a differentiable learning pipeline. To address this issue, following Smirnov et al. (2020), we first define *variational directed Chamfer distance*, starting from equation 2:

$$\mathrm{Ch}_{\text{dir}}(X,Y) = \frac{1}{|X|}\sum_{x\in X}\min_{y\in Y}\mathrm{d}(x,y) \approx \mathbb{E}_{x\sim\mathcal{U}_A}\left[\inf_{y\in Y}\mathrm{d}(x,y)\right] = \frac{1}{\mathrm{Area}(X)}\int_X \inf_{y\in Y}\mathrm{d}(x,y)\mathrm{d}x \stackrel{\text{def}}{=} \mathrm{Ch}^{\text{var}}_{\text{dir}}(A,B), \tag{3}$$

where $\mathcal{U}_A$ is the uniform distribution on A. $\mathrm{Ch}^{\text{var}}(A,B)$ is defined analogously.

While it is difficult to sample uniformly from patches, we can do so easily from their parametric domain (the unit square). We perform a change of variables to get the following (derivation in §A.1):

$$\mathrm{Ch}^{\text{var}}_{\text{dir}}(P,M) = \frac{\mathbb{E}_{(s,t)\sim\mathcal{U}_{[0,1]^2}}\left[\inf_{y\in M}\mathrm{d}(P(s,t),y)|J(s,t)|\right]}{\mathbb{E}_{(s,t)\sim\mathcal{U}_{[0,1]^2}}\left[|J(s,t)|\right]}, \tag{4}$$

where $J(s,t)$ is the Jacobian of $P(s,t)$. We approximate this expression via Monte Carlo integration.

Since we can precompute uniformly sampled random points from the target mesh, we do not need to use area weights for $\mathrm{Ch}^{\text{var}}_{\text{dir}}(M,P)$. Thus, our area-weighted Chamfer distance is

$$\mathcal{L}_{\text{Ch}}(\cup P_i, M) \;=\; \frac{\sum_i\sum_{(s,t)\in U_\square}\min_{y\in M}\mathrm{d}(P(s,t),y)|J_i(s,t)|}{\sum_i\sum_{(s,t)\in U_\square}|J_i(s,t)|} + \frac{\sum_{x\in M}\min_{y\in\cup P_i}\mathrm{d}(x,y)}{|M|}, \tag{5}$$

where $U_\square \sim \mathcal{U}_{[0,1]^2}$. We compute $J_i(u,v)$ for a patch given its control points in closed-form.

Normal alignment. The normal alignment term is computed analogously to Ch_{dir}, except that instead of Euclidean distance, we use $\mathrm{d}_N(x,y) = \|n_x - n_y\|_2^2$, where n_x is the unit normal at x. For each point y sampled from our predicted surface, we compare n_y to n_x, where $x \in M$ is closest to y, and, symmetrically, for each $x' \in M$, we compare $n_{x'}$ to to $n_{y'}$, where $y' \in \cup P_i$ is closest to x':

$$\mathcal{L}_{\text{normal}}(\cup P_i,M) = \frac{\sum_i\sum_{(u,v)\in U_\square}\mathrm{d}_N(\mathrm{NN}_M(P_i(u,v)),P_i(u,v))|J_i(u,v)|}{\sum_i\sum_{(u,v)\in U_\square}|J_i(u,v)|} + \frac{\sum_{x\in M}\mathrm{d}_N(x,\mathrm{NN}_{\cup P_i}(x))}{|M|}, \tag{6}$$

where $\mathrm{NN}_Y(x)$ is the nearest neighbor to x in Y under Euclidean distance.

Intersection regularization. We introduce a *collision detection loss* to detect patch intersections:

$$\mathcal{L}_{\text{coll}}(\{P_i\}) = \sum_{i\neq j}\exp\Big(-\big(\min(\mathrm{d}(\mathcal{T}^i,P^j),\mathrm{d}(\mathcal{T}^j,P^i))/\varepsilon\big)^2\Big), \tag{7}$$

where $\mathcal{T}^i$ is a triangulation of patch P_i, and P^i is a set of points sampled from patch P_i. To triangulate a patch, we take a fixed triangulation of the parameter space (a unit square) and compute the image of each vertex under the Coons patch map, keeping the original connectivity. With a small $\varepsilon = 10^{-6}$, this expression is a smooth indicator for when two patches are intersecting. For a pair of adjacent patches or those that share a junction, we truncate one patch at the adjacency before evaluating the loss.

Patch flatness regularization. To help patches to align to smooth regions and sharp creases to fall on patch boundaries, we define a *patch flatness regularizer*, which discourages excessive curvature by regularizing each Coons patch map $P : [0,1]\times[0,1] \to \mathbb{R}^3$ to be close to a linear map. For each patch, we sample points $U_\square$ in parameter space, compute their image $P(U_\square)$, and fit a linear function using

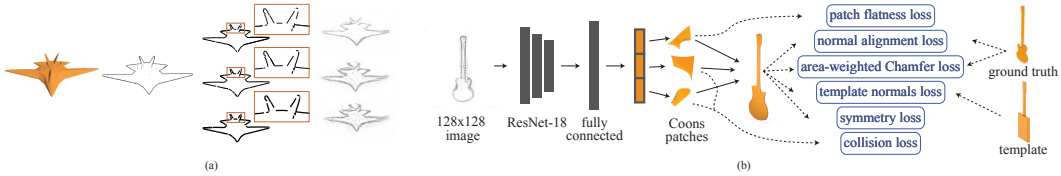


Figure 4: An overview of our data generation and augmentation (a) and learning (b) pipelines.

least-squares. Thus, $\hat{P}(U_\square) = AU_\square + b \approx P(U_\square)$ for some A,b. We define patch flatness loss as

$$\mathcal{L}_{\text{flat}}(\{P_i\}) = \frac{\sum_i \sum_{(u,v)\in U_\square} \|\hat{P}_i(u,v) - P_i(u,v)\|_2^2 |J_i(u,v)|}{\sum_i \sum_{(u,v)\in U_\square} |J_i(u,v)|}. \tag{8}$$

Template normals regularization. For categories where a template is available, we not only initialize the network with the template geometry but also regularize the output using template normals. This favorably positions patch seams and prevents patches from sliding over high-curvature regions:

$$\mathcal{L}_{\text{template}}(\{P_i\},\{T_i\}) = \frac{\sum_i \sum_{(u,v)\in U_\square} \|n_{P_i(u,v)} - n_{T_i}\|_2^2 |J_i(u,v)|}{\sum_i \sum_{(u,v)\in U_\square} |J_i(u,v)|}, \tag{9}$$

where n_{T_i} is the normal vector of the ith template patch.

Global symmetry. Man-made shapes often exhibit global bilateral symmetries. Enforcing symmetry is difficult in, e.g., meshes or implicit surfaces. In contrast, after computing a template's symmetry planes, we enforce symmetric positions of the corresponding control points as an additional loss term:

$$\mathcal{L}_{\text{sym}}(\cup P_i) = \frac{1}{|S|} \sum_{(i,j)\in S} \|(P_x^i - a, P_y^i, P_z^i) - (a - P_x^j, P_y^j, P_z^j)\|_2^2, \tag{10}$$

where S contains index pairs of symmetric control points (P^i, P^j). In the formula, we assume w.l.o.g. symmetry plane $x = a$. We use this to enforce symmetrical reconstruction of airplanes and cars.

3.3 Deep Learning Pipeline

The final loss that we optimize is

$$\begin{aligned}\mathcal{L}(\{P_i\},M) = {} & \mathcal{L}_{\text{Ch}}(\cup P_i M) + \alpha_{\text{normal}} \mathcal{L}_{\text{normal}}(\cup P_i M) \\ & + \alpha_{\text{flat}} \mathcal{L}_{\text{flat}}(\{P_i\}) + \alpha_{\text{coll}} \mathcal{L}_{\text{coll}}(\{P_i\}) + \alpha_{\text{template}} \mathcal{L}_{\text{template}}(\{P_i\},\{T_i\}) + \alpha_{\text{sym}} \mathcal{L}_{\text{sym}}(\cup P_i).\end{aligned} \tag{11}$$

For models scaled to fit in a unit sphere, we use $\alpha_{\text{normal}} = 0.008$, $\alpha_{\text{flat}} = 2$, and $\alpha_{\text{coll}} = 0.00001$ for all experiments, and $\alpha_{\text{template}} = 0.0001$ and $\alpha_{\text{sym}} = 1$ for experiments that use those regularizers.

Our network inputs one or more 128×128 images and outputs patch parameters. The architecture is ResNet-18 (He et al., 2016) followed by hidden layers with 1024, 512, and 256 units, and an output layer with size equal to the output dimension. Final layer weights are initialized to zero with bias equal to the template parameters. For multi-view input, we encode each image and do max pooling over the latent codes. We use ReLU and batch normalization after each layer except the last. We train each network for 24 hours on a Tesla V100 GPU, using Adam (Kingma & Ba, 2014) and batch size 8 with learning rate 0.0001 for sphere templates and 0.0001 for category-specific templates. At each iteration, we sample 7,000 points from the predicted and target shapes. Our pipeline is illustrated in Fig. 4b.

4 Experimental Results

We introduce a pipeline for generating realistic sketch data and train a network that coverts a sketch image to a patch-based 3D representation. We show results on synthetic and human-drawn sketches, demonstrate interpolation and quad meshing, compare to prior work, and do an ablation study (§A.2).

4.1 Data Preparation

While there exist annotated datasets of 3D models and corresponding hand-drawn sketches (Gryaditskaya et al., 2019), such data are unavailable at the deep learning scale. Instead, we generate synthetic data. Guided by Cole et al. (2012), we first render occluding contours and sharp edges using the Arnold Toon Shader in Autodesk Maya for each model from representative camera views. Although the contour images capture the main features of the 3D model, they lack some of the ambiguities of

Figure 5: Results on synthetic sketches. For each category, from top to bottom: input sketch, output 3D model with sphere template (54 patches), output 3D model with category-specific template.

rough hand-drawn sketches (Liu et al., 2018). To this end, we vectorize the images using Bessmeltsev & Solomon (2019) and augment the set of contours by stochastically splitting or truncating curves. Finally, we rasterize each contour image using different stroke widths and pass it through the pencil drawing generation model of Simo-Serra et al. (2018). We illustrate this pipeline in Fig. 4a.

Thus, we obtain realistic sketch images paired with 3D models. We train on the airplane, bathtub, guitar, bottle, car, mug, gun, and knife categories of ShapeNet Core (v2) (Chang et al., 2015). We make the meshes watertight using Huang et al. (2018) and normalize them to fit an origin-centered unit sphere.

4.2 Real and Synthetic Sketch Reconstruction

We pick a random 10%-90% test-train split for each category and evaluate in Fig. 5 as well as §A.5.

The templates for airplanes, guitars, guns, knives, and cars are generated automatically using segmentations of Yi et al. (2016). For mugs, we start with an automatic template and add a hole in the handle and a void in the interior. To demonstrate a template with distinct parts, for cars, we use the segmentation during training, computing Chamfer and normal losses for wheels and body separately. For bottle and bathtub templates, we simply place two and and five cuboids, respectively.

With a generic sphere template, we produce a compact piecewise-smooth surface of comparable quality to the more conventional deformable meshes. Our algorithmic construction of category-specific templates, however, enables higher-quality reconstruction of sharp features and details.

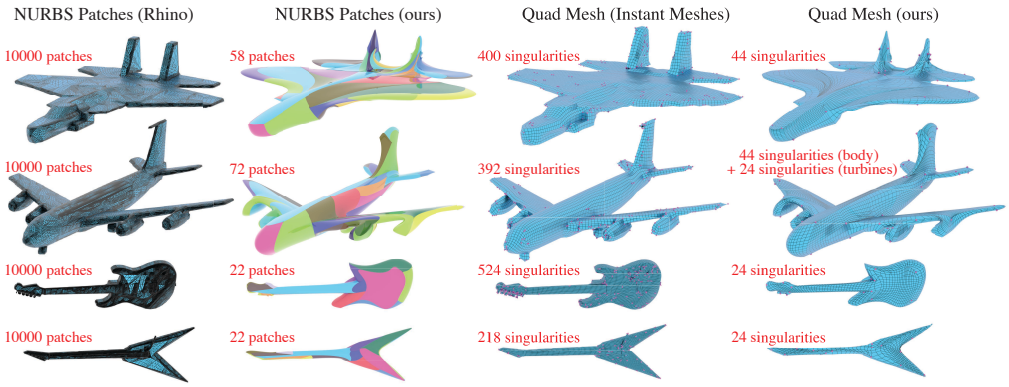
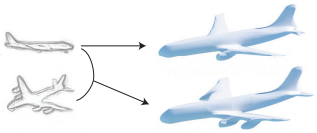
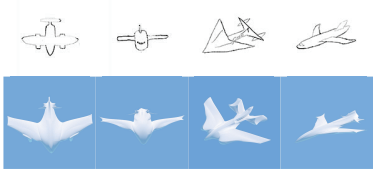


Figure 6: We convert two airplane and two guitar models to NURBS patches using Rhino 3D and compare to our Coons patches. Corresponding Coons patches across models of the same category are the same color. We also show quad meshes generated using Instant Meshes (Jakob et al., 2015) and those from our patch decompositions. Singular points are in pink. Our representation is much more compact and hence easily editable. We produce fewer singularities and only in known places. Furthermore, unlike with other methods, our patches and quad meshes are consistent across models.

We demonstrate our method's ability to incorporate details from different views. We show our output when given a single view of an airplane as well as when given an additional view. The combined model incorporates elements not visible in the original view.

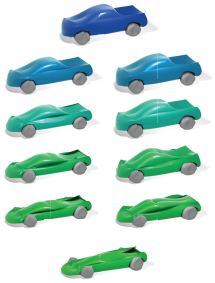


We also test on real sketches. Each artist was asked to sketch an airplane without having seen any sketches from our dataset. The results are similar to those on synthetic sketches, demonstrating that our dataset is reflective of choices that humans make when sketching 3D objects.



4.3 3D Model Interpolation

Our representation is naturally well-suited for interpolating between 3D models. As each model is composed of a small number of consistently placed patches, we linearly interpolate the patch parameters (e.g., vertex positions) to generate models "between" our network outputs. We also perform interpolation in the latent space learned by our model (the output of the first 1024-dimensional hidden layer). While latent-space interpolation is similar to patch-space, each interpolant better resembles a realistic model due to the learned priors. We demonstrate patch-space (right) and latent-space (left) interpolation between two car models.



4.4 Quad Meshing and NURBS Decomposition

Many algorithms have been designed for converting triangle meshes into NURBS patches (Krishnamurthy & Levoy, 1996; Eck & Hoppe, 1996; Bernardini et al.). By fitting a template to a 3D model, our method automatically converts the model to a set of Coons patches. Moreover, our Coons patches are placed *consistently*, thus establishing correspondences between models. In Fig. 6, we compare our Coons patches to NURBS patches automatically obtained using Rhino 3D, a commercial CAD program. Our decomposition is significantly more sparse and usable for further editing.

A common task in computer graphics is converting surfaces into quad meshes. We can easily obtain a quad mesh from our patch decomposition by simply subdividing (see §A.4 for details). We compare our quad meshes to those produced by Instant Meshes (Jakob et al., 2015), a recent quad meshing algorithm, in Fig. 4.4. It is commonly desired to minimize the number of singular points (vertices with valence not equal to four) and to consistently position them in a quad mesh. Our algorithm achieves both of these goals—the number and location of singularities is determined by our template. While the Instant Meshes results contain between 215 and 524 singular points, our quad meshes contain only 24 (guitars), 44 (airplane without turbines), or 68 (airplane with turbines).

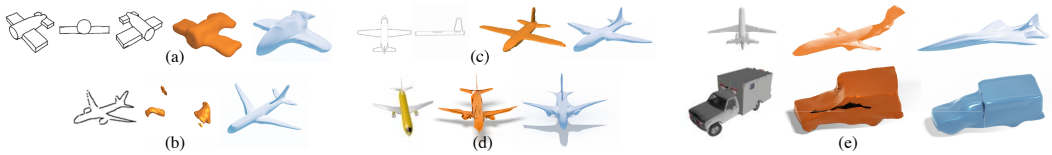


Figure 7: Comparisons to (Delanoy et al., 2018) using their data (a) and our data (b), to (Lun et al., 2017) (c), to AtlasNet (d), and to Pixel2Mesh (e). Our results are in blue for each comparison.

4.5 Comparisons

In Fig. 7, we compare to the sketch-based reconstruction methods of Delanoy et al. (2018) (a) and Lun et al. (2017) (c). Although we use the same species of input used to train these methods rather than attempting to re-train their models on our data, the visual quality of our predictions is comparable to theirs. Moreover, our output representation sparsely captures smooth and sharp features, independent of resolution. In contrast, Delanoy et al. (2018) produce a 64^3 voxel grid—a dense, fixed-resolution representation, which cannot be edited directly and offers no topological guarantees. In Fig. 7b, we evaluate their system on contours from our dataset, demonstrating that our task of reconstruction with a prior on class (airplane) rather than structure (cylinders and cuboids) is misaligned with theirs: Since our data is not well-approximated by CSG models, their method cannot extract meaningful output.

Although Lun et al. (2017) ultimately produce a mesh, they perform a computationally expensive post-processing procedure, since their forward pass returns a labeled point cloud. Our method directly outputs the parameters for surface patches with no further optimization. Additionally, their final mesh contains more components (triangles) than our output (patches), making it less useful for editing.

In Fig. 7d, we compare to AtlasNet (Groueix et al., 2018). We retrain our model with their renderings, using the 54-face sphere template. While our 3D reconstructions capture the same degree of detail, they do not suffer from topological defects. In particular, AtlasNet's surfaces contains many patch intersections and holes. Extracting a watertight mesh would require significant post-processing. Additionally, each patch in our representation is parameterized sparsely by control points, in contrast to AtlasNet's patches, which must be sampled using a deep decoder network.

In Pixel2Mesh, Wang et al. (2018b) input an image and output a triangle mesh. We train our sphere template models on their data. While their meshes have 2466 vertices (*7398 degrees of freedom*) we output 54 patches (*816 degrees of freedom*)—a more editable and interpretable representation. As shown in Fig. 7e, the low dimensionality of our output is not at the expense of expressiveness.

5 Discussion and Conclusion

As 3D data become more readily available, the need for processing, modifying, and generating 3D models in a usable fashion increases. While the quality of results produced by deep learning systems improves, it is necessary to think carefully about their format, particularly with respect to existing use cases. By carefully designing representations together with compatible learning algorithms, we can harness these data for the purpose of simplifying workflows in design, modeling, and manufacturing.

Our system is a step toward practical 3D modeling assisted by deep learning. Our sparse patch-based representation is close to what is used in artistic and engineering practice, and we accompany it with new geometric regularizers that greatly improve the reconstruction process. Unlike meshes or voxel occupancy functions, this representation can easily be edited and tuned after 3D reconstruction, and it captures a trade-off between smoothness and sharp edges reasonable for man-made shapes.

Our work suggests several avenues for future research. While we use pre-trained networks to generate sketch data, we could couple the training of these pieces to alleviate dependence on sketch–3D model pairs. We also could leverage literature in computer-aided geometric design to identify other structures amenable to learning. Particularly, multiresolution representations (e.g., subdivision surfaces) might enable learning both high-level smooth structure and geometric details like filigree independently.

Perhaps the most important challenge remaining from our work—and others—involves inferring shape *topology*. While our system supports structural variability and modular parts, scaling this towards completely learned topology is nontrivial. This limitation is reasonable for shape classes we consider, but generic sketch reconstruction requires automatically adding and connecting patches.

REFERENCES

Timur Bagautdinov, Chenglei Wu, Jason Saragih, Pascal Fua, and Yaser Sheikh. Modeling facial geometry using compositional vaes. In *CVPR*, June 2018.

Pierre Baque, Edoardo Remelli, François Fleuret, and Pascal Fua. Geodesic convolutional shape optimization. *arXiv preprint arXiv:1802.04016*, 2018.

Fausto Bernardini, Chandrajit L Bajaj, Jindong Chen, and Daniel R Schikore. Automatic reconstruction of 3d cad models from digital scans. *International Journal of Computational Geometry & Applications*, 9(04n05):327–369.

Mikhail Bessmeltsev and Justin Solomon. Vectorizing Line Drawings via Polyvector Fields. *ACM TOG*, 38(1), 2019.

Mikhail Bessmeltsev, Will Chang, Nicholas Vining, Alla Sheffer, and Karan Singh. Modeling character canvases from cartoon drawings. *ACM Trans. Graph.*, 34(5):162:1–162:16, November 2015.

Mikhail Bessmeltsev, Nicholas Vining, and Alla Sheffer. Gesture3D: Posing 3D Characters via Gesture Drawings. *ACM TOG*, 35(6):165:1–165:13, 2016.

Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical report, Stanford University – Princeton University – Toyota Technological Institute at Chicago, 2015.

Tao Chen, Zhe Zhu, Ariel Shamir, Shi-Min Hu, and Daniel Cohen-Or. 3-Sweep. *ACM TOG*, 32(6): 1–10, 2013.

Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *CVPR*, pp. 5939–5948, 2019.

Joseph Jacob Cherlin, Faramarz Samavati, Mario Costa Sousa, and Joaquim A. Jorge. Sketch-based modeling with few strokes. *SCCG*, 1(212):137, 2005.

Forrester Cole, Aleksey Golovinskiy, Alex Limpaecher, Heather Stoddart Barros, Adam Finkelstein, Thomas Funkhouser, and Szymon Rusinkiewicz. Where do people draw lines? *Communications of the ACM*, 55(1):107, 2012.

S. A. Coons. Surfaces for computer-aided design of space forms. Technical report, 1967.

Johanna Delanoy, Mathieu Aubry, Phillip Isola, Alexei A Efros, and Adrien Bousseau. 3d sketching using multi-view deep volumetric prediction. *Conference on Computer Graphics and Interactive Techniques*, 1(1):1–22, 2018.

Zhantao Deng, Jan Bednařík, Mathieu Salzmann, and Pascal Fua. Better patch stitching for parametric surface reconstruction. *arXiv preprint arXiv:2010.07021*, 2020.

Chao Ding and Ligang Liu. A survey of sketch based modeling systems. *Front. Comput. Sci.*, 10(6): 985–999, December 2016.

Matthias Eck and Hugues Hoppe. Automatic reconstruction of b-spline surfaces of arbitrary topological type. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 325–334, 1996.

Diaa F Elkott, Hoda A Elmaraghy, and Waguih H Elmaraghy. Automatic sampling for cmm inspection planning of free-form surfaces. *international Journal of production Research*, 40(11): 2653–2676, 2002.

Even Entem, Loic Barthe, Marie-Paule Cani, Frederic Cordier, and Michiel van de Panne. Modeling 3d animals from a side-view sketch. *Comput. Graph.*, 46(C):221–230, February 2015.

Haoqiang Fan, Hao Su, and Leonidas Guibas. DeepPointSet : A Point Set Generation Network for 3D Object Reconstruction from a Single Image. In *CVPR*, 2017.

Jun Gao, Chengcheng Tang, Vignesh Ganapathi-Subramanian, Jiahui Huang, Hao Su, and Leonidas J. Guibas. DeepSpline: Data-Driven Reconstruction of Parametric Curves and Surfaces. pp. 1–13, 2019.

Kyle Genova, Forrester Cole, Daniel Vlasic, Aaron Sarna, William T Freeman, and Thomas Funkhouser. Learning shape templates with structured implicit functions. In *ICCV*, pp. 7154–7164, 2019.

Yotam Gingold, Takeo Igarashi, and Denis Zorin. Structured annotations for 2D-to-3D modeling. *ACM TOG*, 28(5):1, 2009.

Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan C. Russell, and Mathieu Aubry. AtlasNet: A Papier-Mache Approach to Learning 3D Surface Generation. *CVPR*, pp. 216–224, 2018.

Yulia Gryaditskaya, Mark Sypesteyn, Jan Willem Hoftijzer, Sylvia Pont, Frédo Durand, and Adrien Bousseau. Opensketch: A richly-annotated dataset of product design sketches. *ACM TOG*, 38, 11 2019.

Christian Häne, Shubham Tulsiani, and Jitendra Malik. Hierarchical surface prediction. *TPAMI*, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pp. 770–778, 2016.

Victoria Hernández-Mederos and Jorge Estrada-Sarlabous. Sampling points on regular parametric curves with control of their distribution. *Computer Aided Geometric Design*, 20(6):363–382, 2003.

H. Huang, E. Kalogerakis, E. Yumer, and R. Mech. Shape synthesis from sketches via procedural models and convolutional networks. *IEEE TVCG*, 23(8):2003–2013, Aug 2017.

Jingwei Huang, Hao Su, and Leonidas Guibas. Robust watertight manifold surface generation method for shapenet models. *arXiv preprint arXiv:1802.01698*, 2018.

Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A sketching interface for 3d freeform design. In *Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pp. 409–416, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co. ISBN 0-201-48560-5.

Anil K. Jain, Yu Zhong, and Marie-Pierre Dubuisson-Jolly. Deformable template models: A review. *Signal Process.*, 71(2):109–129, December 1998.

Wenzel Jakob, Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. Instant field-aligned meshes. *ACM Transactions on Graphics (Proceedings of SIGGRAPH ASIA)*, 34(6), November 2015. doi: 10.1145/2816795.2818078.

Angjoo Kanazawa, Shubham Tulsiani, Alexei A. Efros, and Jitendra Malik. Learning category-specific mesh reconstruction from image collections. In *ECCV*, 2018.

Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In *CVPR*, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Venkat Krishnamurthy and Marc Levoy. Fitting smooth surfaces to dense polygon meshes. In *Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pp. 313–324, New York, NY, USA, 1996. Association for Computing Machinery. ISBN 0897917464.

Changjian Li, Hao Pan, Yang Liu, Xin Tong, Alla Sheffer, and Wenping Wang. Robust flow-guided neural prediction for sketch-based freeform surface modeling. *ACM Trans. Graph.*, 37(6), December 2018.

Or Litany, Alex Bronstein, Michael Bronstein, and Ameesh Makadia. Deformable shape completion with graph convolutional autoencoders. *CVPR*, 2018.

Chenxi Liu, Enrique Rosales, and Alla Sheffer. Strokeaggregator: Consolidating raw sketches into artist-intended curve drawings. *ACM TOG*, 37(4), 2018.

M. Liu, O. Tuzel, A. Veeraraghavan, and R. Chellappa. Fast directional chamfer matching. In *CVPR*, pp. 1696–1703, June 2010.

Zhaoliang Lun, Matheus Gadelha, Evangelos Kalogerakis, Subhransu Maji, and Rui Wang. 3d shape reconstruction from sketches via multi-view convolutional networks. In *3DV*, 2017.

Priyanka Mandikal, K L Navaneet, Mayank Agarwal, and R Venkatesh Babu. 3D-LMNet: Latent embedding matching for accurate and diverse 3d point cloud reconstruction from a single image. In *BMVC*, 2018.

Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *CVPR*, 2019.

Charlie Nash, Yaroslav Ganin, SM Eslami, and Peter W Battaglia. Polygen: An autoregressive generative model of 3d meshes. *arXiv preprint arXiv:2002.10880*, 2020.

Gen Nishida, Ignacio Garcia-Dorado, Daniel G. Aliaga, Bedrich Benes, and Adrien Bousseau. Interactive sketching of urban procedural models. *ACM Trans. Graph.*, 35(4), 2016.

Luke Olsen, Faramarz F. Samavati, Mario Costa Sousa, and Joaquim A. Jorge. Sketch-based modeling: A survey. *Computers & Graphics*, 33(1):85–103, feb 2009.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *CVPR*, June 2019.

Les Piegl and Wayne Tiller. *The NURBS Book*. Springer-Verlag, New York, NY, USA, second edition, 1996.

Danilo Jimenez Rezende, SM Ali Eslami, Shakir Mohamed, Peter Battaglia, Max Jaderberg, and Nicolas Heess. Unsupervised learning of 3d structure from images. In *Advances in neural information processing systems*, pp. 4996–5004, 2016.

Edgar Simo-Serra, Satoshi Iizuka, and Hiroshi Ishikawa. Mastering Sketching: Adversarial Augmentation for Structured Prediction. *ACM TOG*, 37(1), 2018.

Dmitriy Smirnov, Matthew Fisher, Vladimir G. Kim, Richard Zhang, and Justin Solomon. Deep parametric shape predictions using distance fields. In *CVPR*, 2020.

Chunyu Sun, Qianfang Zou, Xin Tong, and Yang Liu. Learning adaptive hierarchical cuboid abstractions of 3d shape collections. *ACM TOG*, 38(6), 2019.

Shubham Tulsiani, Hao Su, Leonidas J. Guibas, Alexei A. Efros, and Jitendra Malik. Learning shape abstractions by assembling volumetric primitives. In *CVPR*, 2017a.

Shubham Tulsiani, Tinghui Zhou, Alexei A. Efros, and Jitendra Malik. Multi-view Supervision for Single-view Reconstruction via Differentiable Ray Consistency. 2017b.

Shubham Tulsiani, Alexei A Efros, and Jitendra Malik. Multi-view consistency as supervisory signal for learning shape and pose prediction. In *CVPR*, pp. 2897–2905, 2018.

Jian Wang, Xiang Jiang, LA Blunt, Richard K Leach, and Paul J Scott. Intelligent sampling for the measurement of structured surfaces. *Measurement Science and Technology*, 23(8):085006, 2012.

Lingjing Wang, Jifei Wang, Cheng Qian, and Yi Fang. Unsupervised learning of 3d model reconstruction from hand-drawn sketches. In *ACM MM*, ACM MM, pp. 1820–1828. Association for Computing Machinery, Inc, 2018a. 26th ACM Multimedia conference, MM 2018 ; Conference date: 22-10-2018 Through 26-10-2018.

Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *ECCV*, 2018b.

Weiyue Wang, Duygu Ceylan, Radomir Mech, and Ulrich Neumann. 3dn: 3d deformation network. In *CVPR*, 2019.

Francis Williams, Teseo Schneider, Claudio Silva, Denis Zorin, Joan Bruna, and Daniele Panozzo. Deep geometric prior for surface reconstruction. In *CVPR*, pp. 10130–10139, 2019.

Jiajun Wu, Tianfan Xue, Joseph J Lim, Yuandong Tian, Joshua B Tenenbaum, Antonio Torralba, and William T Freeman. Single image 3d interpreter network. In *ECCV*, 2016.

Jiajun Wu, Yifan Wang, Tianfan Xue, Xingyuan Sun, William T Freeman, and Joshua B Tenenbaum. MarrNet: 3D Shape Reconstruction via 2.5D Sketches. In *Advances In Neural Information Processing Systems*, 2017.

Jiajun Wu, Chengkai Zhang, Xiuming Zhang, Zhoutong Zhang, William T Freeman, and Joshua B Tenenbaum. Learning 3D Shape Priors for Shape Completion and Reconstruction. In *ECCV*, 2018.

Xinchen Yan, Jimei Yang, Ersin Yumer, Yijie Guo, and Honglak Lee. Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision. In *Advances in neural information processing systems*, pp. 1696–1704, 2016.

Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. Foldingnet: Point cloud auto-encoder via deep grid deformation. In *CVPR*, June 2018.

Li Yi, Vladimir G. Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. A scalable active framework for region annotation in 3d shape collections. *SIGGRAPH Asia*, 2016.

Kangxue Yin, Hui Huang, Daniel Cohen-Or, and Hao Zhang. P2p-net: Bidirectional point displacement net for shape transform. *ACM Transactions on Graphics(Special Issue of SIGGRAPH)*, 37 (4):152:1–152:13, 2018.

Mehmet Ersin Yumer and Levent Burak Kara. Surface creation on unstructured point sets using neural networks. *Computer-Aided Design*, 44(7):644 – 656, 2012.

Xiuming Zhang, Zhoutong Zhang, Chengkai Zhang, Joshua B Tenenbaum, William T Freeman, and Jiajun Wu. Learning to Reconstruct Shapes from Unseen Classes. In *NeurIPS*, 2018.

A APPENDIX

A.1 CHANGE OF VARIABLES FOR VARIATIONAL CHAMFER DISTANCE

We derive the expression for area-weighted variational Chamfer distance:

$$\mathrm{Ch}^{\mathrm{var}}_{\mathrm{dir}}(P,M) = \frac{1}{\mathrm{Area}(P)} \int_P \inf_{y \in M} \mathrm{d}(x,y) \mathrm{d}x \tag{12}$$

$$= \frac{1}{\mathrm{Area}(P)} \int_0^1 \int_0^1 \inf_{y \in M} \mathrm{d}(P(s,t),y) |J(s,t)| \mathrm{d}s\mathrm{d}t \tag{13}$$

$$= \frac{1}{\mathrm{Area}(P)} \cdot \frac{1}{\mathrm{Area}(\square)} \int_0^1 \int_0^1 \inf_{y \in M} \mathrm{d}(P(s,t),y) |J(s,t)| \mathrm{d}s\mathrm{d}t \tag{14}$$

$$= \frac{1}{\mathrm{Area}(P)} \mathbb{E}_{(s,t) \sim \mathcal{U}_\square} \left[\inf_{y \in M} \mathrm{d}(P(s,t),y) |J(s,t)| \right] \tag{15}$$

$$= \frac{\mathbb{E}_{(s,t) \sim \mathcal{U}_\square} [\inf_{y \in M} \mathrm{d}(P(s,t),y) |J(s,t)|]}{\mathbb{E}_{(s,t) \sim \mathcal{U}_\square} [|J(s,t)|]} \tag{16}$$

A.2 ABLATION STUDY

We perform an ablation study of our method on an airplane model, demonstrating the effect of training without each term in our loss function as well as the difference between a category-specific template, a 54-patch sphere template, and a lower resolution 24-patch template. The results are shown in Fig. 8. We also show an ablation results on the knife model in Fig. 9.

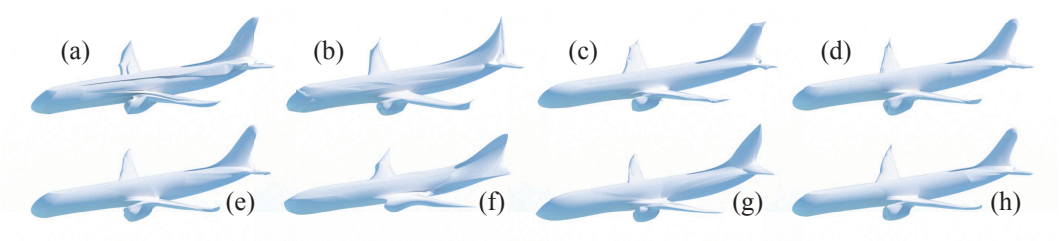


Figure 8: An ablation study of our model without normal alignment loss (a), without collision detection loss (b), without patch flatness loss (c), without template normal loss (d), without symmetry loss (e), as well as using 24-patch (f) and 54-patch (g) sphere templates compared to the final result (h).

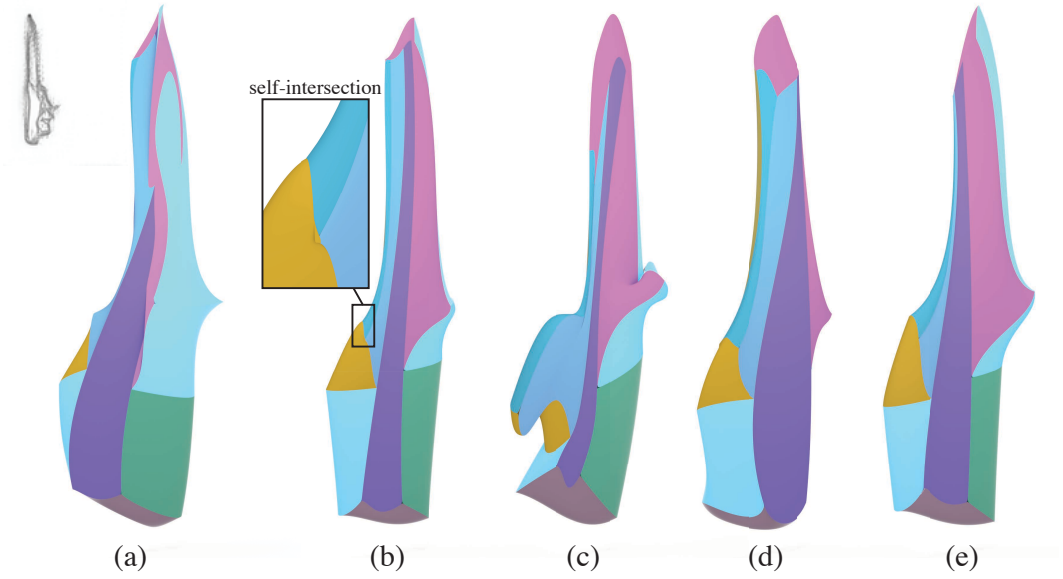


Figure 9: An ablation study of our model without normal alignment loss (a), without collision detection loss (b), without patch flatness loss (c), without template normal loss (d), compared to the final result (e).

The ablation study demonstrates the contribution of each component of our system method to the final result. Training without collision detection loss results in patch intersections. Omitting the normal loss causes the 3D surface to suffer in smoothness. Patch flatness and and template normal losses encourage patch seams to align to sharp features. While both sphere templates capture the geometry, using more patches captures greater details, and using a non-generic template further improves the model.

We additionally provide a quantitative ablation study for the knives and airplanes category in Table 1. We train a model without each of the normals, flatness, template normals, and collision loss terms for 50000 steps. for each model, we report the average Chamfer loss, normals loss, flatness loss, and template normals loss (as defined in §3.2) on the test set, in addition to the average number of intersecting patch pairs. The results demonstrate that each loss term contributes to our full model without sacrificing reconstruction quality.

Model	Ch. Loss	Norm. Loss	Flat. Loss	Temp. Norm. Loss	# Int. Pairs
airplanes (full)	0.0118	0.725	0.0000732	0.792	0.0220
airplanes (no norm.)	0.0105	1.09	0.0000941	0.935	0.0357
airplanes (no flat.)	0.0119	0.748	0.000254	0.866	0.0302
airplanes (no temp. norm.)	0.0114	0.723	0.0000734	0.889	0.0110
airplanes (no coll.)	0.0112	0.727	0.000113	1.07	3.09
knives (full)	0.0124	0.635	0.0000950	0.669	0
knives (no norm.)	0.0124	1.12	0.0000902	1.46	3.95
knives (no flat.)	0.0122	0.648	0.00132	0.733	0.0263
knives (no temp. norm.)	0.0126	0.671	0.0000886	1.03	0.132
knives (no coll.)	0.0122	0.631	0.0000963	0.722	0.0263

Table 1: Quantitative ablation study of our loss function.

A.3 QUANTITATIVE COMPARISON TO PIXEL2MESH

We compare to Pixel2Mesh quantitatively in Table 2. We select 2500 random test set views and compute Chamfer distance using 5000 sampled points. While obtain comparable Chamfer distance values, our representation is significantly more compact, editable, and less prone to non-manifold artifacts.

Category	CD		DOF	
	P2M	ours	P2M	ours
airplane	0.022	0.025	7398	816
car	0.018	0.022	7398	816

Table 2: Quantitative comparison to Pixel2Mesh. We report Chamfer distance (CD) and degrees of freedom in the representation (DOF). We obtain comparable Chamfer distance using a representation that is an order of magnitude more compact and without non-manifold artifacts.

A.4 QUAD MESH GENERATION

Given a collection of Coons patches, we obtain a quad mesh by subdividing each patch into some number of quads. In order for the quad mesh to be uniform, we determine the number of subdivisions for each boundary curve of each patch by solving an integer linear problem (ILP). Our free variables are the number of subdivisions per curve, and the objective function encourages the number of subdivisions to be proportional to the arc length of each curve. The constraints ensure that opposite curves for each patch are subdivided into an equal number of segments, and when one curve contains multiple other curves (due to edge junctions), the numbers of subdivisions are compatible. In particular, we solve the following ILP:

$$\begin{aligned} \text{minimize} \quad & \sum_{i \in I} (x_i - \alpha s_i)^2 \\ \text{s.t.} \quad & x_i = x_{\mathcal{O}(i)} && \forall i \in \mathcal{E} \\ & x_i = \sum_{j \in \mathcal{J}(i)} x_j && \forall i \in \mathcal{E} \\ & x_i \in \mathbb{Z}^+ && \forall i \in \mathcal{E}, \end{aligned}$$

where x_i is the number of subdivisions for curve i, s_i is the arc length of curve i, $\mathcal{E}$ contains all of the curves, $\mathcal{O}$ maps a curve to the opposite curve in its patch, $\mathcal{J}$ maps a curve to a list of curves contained within it, and α is a parameter that controls the coarseness of the resulting quad mesh.

We solve the ILP using MOSEK, subdivide our Coons patches to obtain a quad mesh, and then perform surface-preserving Laplacian smoothing in MeshLab for 10 iterations with maximum angle displacement of $5°$.

A.5 ADDITIONAL QUALITATIVE RESULTS

We show models from the test set for each shape category produced by our full model trained with category-specific templates.

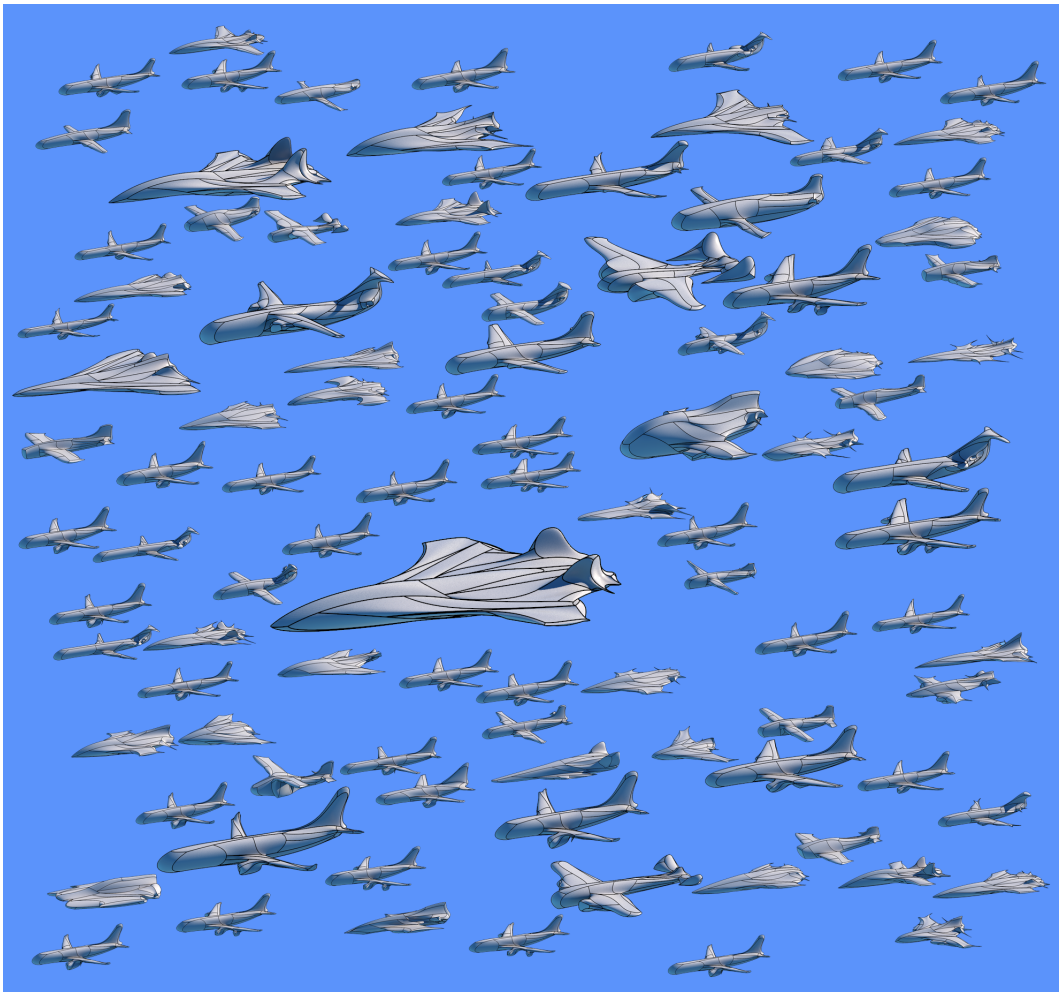


Figure 10: Airplanes from the test set.

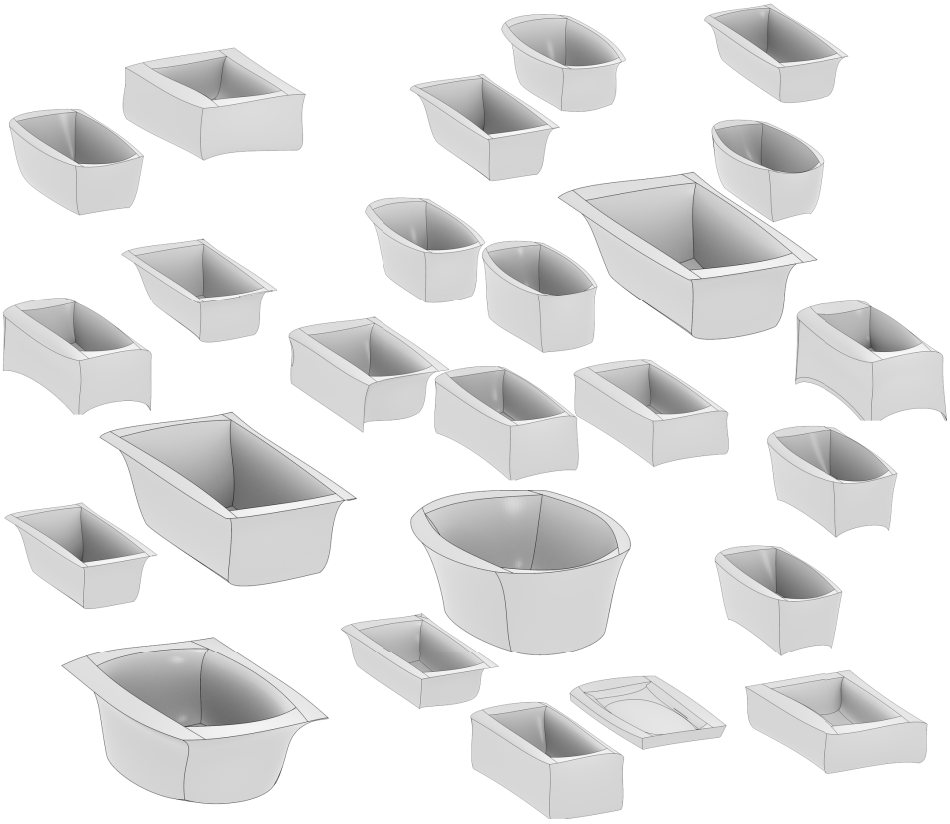


Figure 11: Bathtubs from the test set.

Figure 12: Bottles from the test set.

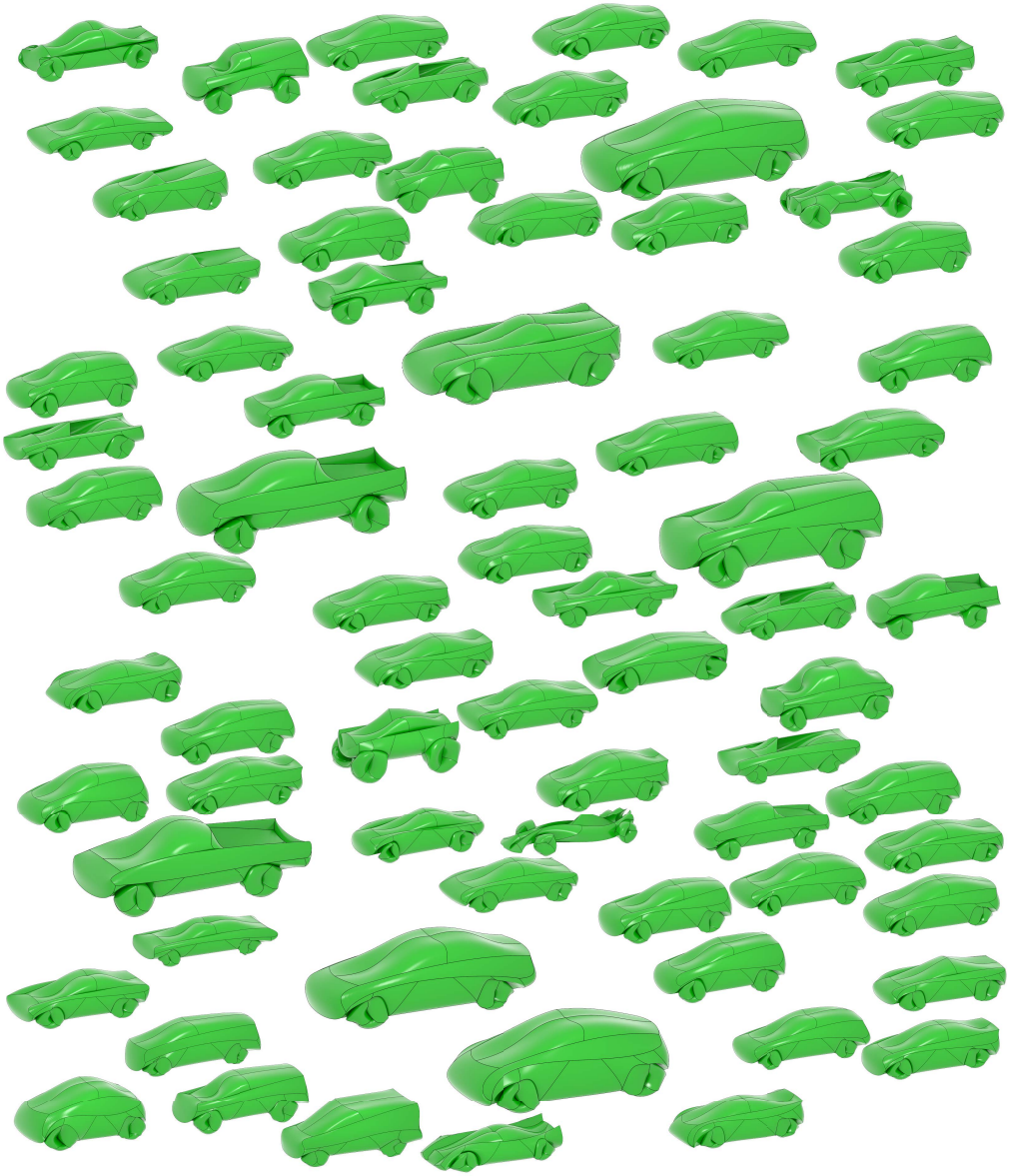


Figure 13: Cars from the test set.

Figure 14: Guitars from the test set.

Figure 15: Guns from the test set.

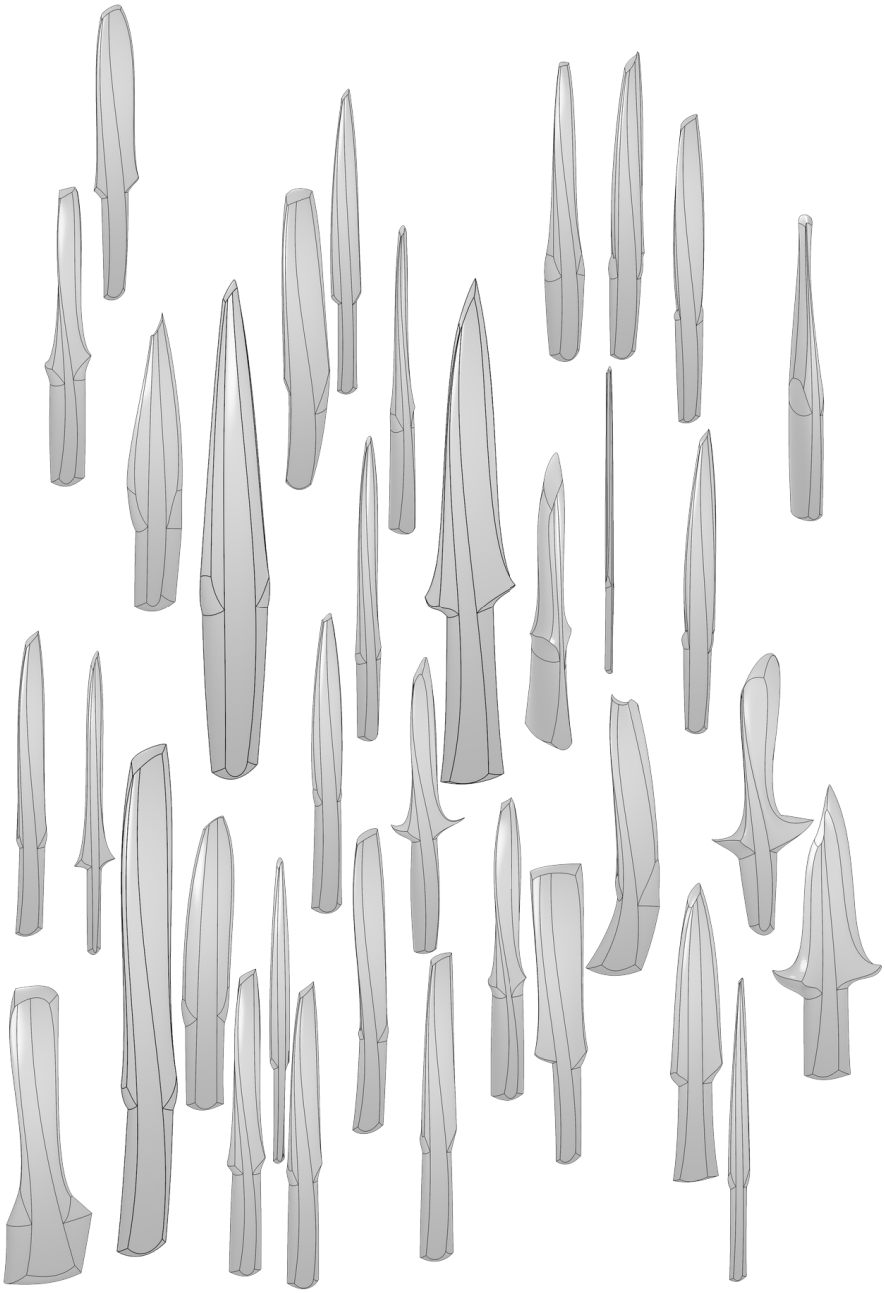


Figure 16: Knives from the test set.

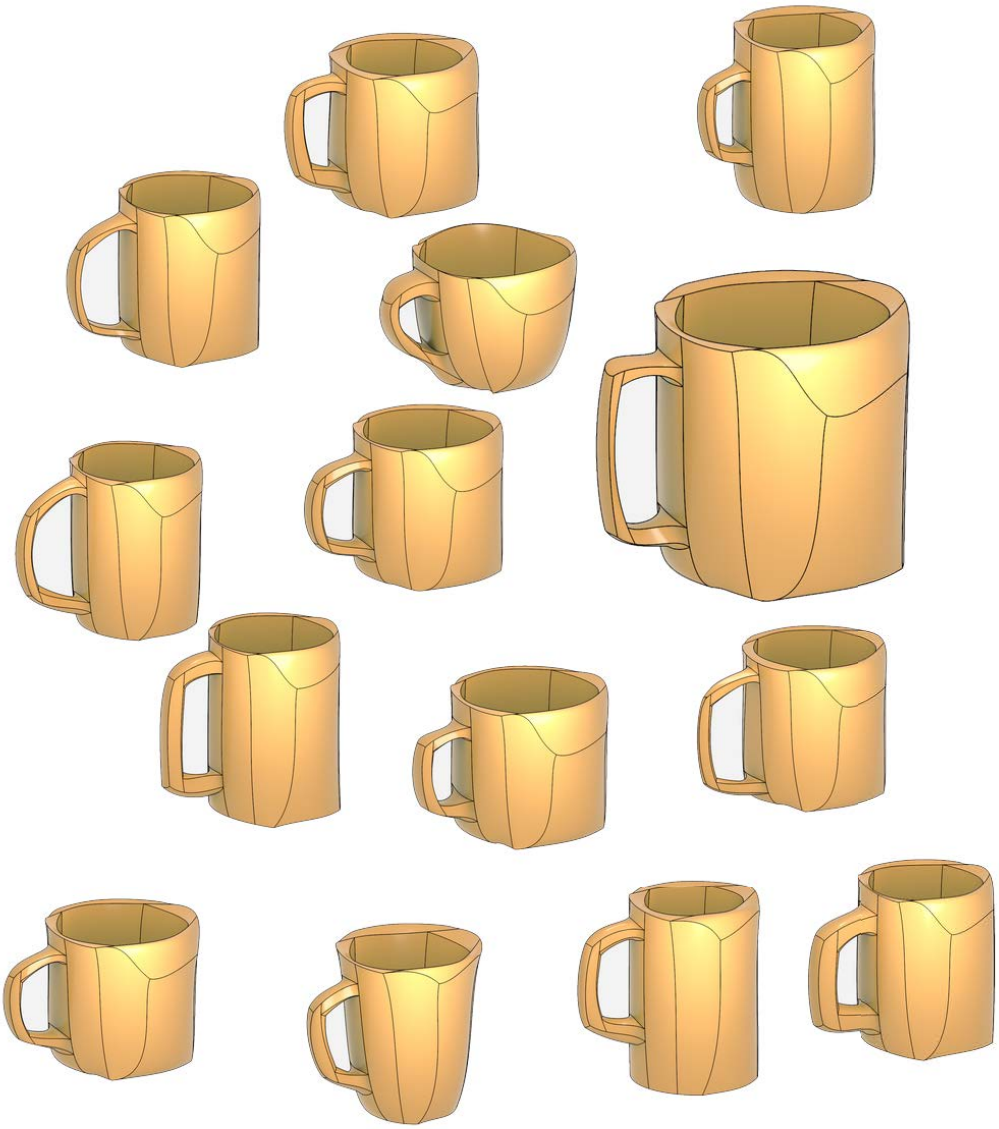


Figure 17: Mugs from the test set.