# 3D Shape Reconstruction from Sketches via Multi-view Convolutional Networks

Zhaoliang Lun  Matheus Gadelha  Evangelos Kalogerakis  Subhransu Maji  Rui Wang
University of Massachusetts Amherst

## Abstract

*We propose a method for reconstructing 3D shapes from 2D sketches in the form of line drawings. Our method takes as input a single sketch, or multiple sketches, and outputs a dense point cloud representing a 3D reconstruction of the input sketch(es). The point cloud is then converted into a polygon mesh. At the heart of our method lies a deep, encoder-decoder network. The encoder converts the sketch into a compact representation encoding shape information. The decoder converts this representation into depth and normal maps capturing the underlying surface from several output viewpoints. The multi-view maps are then consolidated into a 3D point cloud by solving an optimization problem that fuses depth and normals across all viewpoints. Based on our experiments, compared to other methods, such as volumetric networks, our architecture offers several advantages, including more faithful reconstruction, higher output surface resolution, better preservation of topology and shape structure.*

## 1. Introduction

We consider the problem of 3D shape reconstruction from sketches. Contours in a sketch convey important characteristics of the underlying shape such as its figure-ground boundaries, surface curvature, and occlusions [31, 60, 37]. They are also commonly used by artists in the initial stages of character design and object modeling due to the relative ease of sketching. However, the process of converting sketches to a 3D model is time consuming and cumbersome.

We propose an architecture to infer a 3D shape that is consistent with sketches from one or more views of an object. Our method is based on a Convolutional Network (ConvNet) trained to map sketches to 3D shapes. Although ConvNets have been successfully applied to a number of image modality transformation tasks [33, 70, 27, 59, 26], their use for explicit 3D shape generation poses numerous challenges. Most prior work has used voxel-based representations for 3D shapes [63, 7, 67]. However, this scales poorly with the resolution of the voxel grid. 3D shapes can be instead efficiently represented through surface-based representations, such as polygon meshes. However, it is difficult to parameterize meshes in a consistent manner such that they are generated by ConvNets, and unlike voxels, they are not amenable to convolutions over regular grids. Thus, their applicability has been limited to categories (*e.g*., faces, human bodies) where surface elements can be consistently parameterized through correspondence techniques and generated through simple generative models [2, 4, 5, 23].

In this work we instead adopt a multi-view architecture for 3D shape reconstruction inspired by recent work showing that ConvNets have the ability to model geometric and viewpoint transformations of an object given natural images [11, 55, 56, 68, 71]. However, unlike prior multi-view synthesis works, we consider the *full pipeline* of 3D shape reconstruction, and also condition it on line drawings, which are more challenging inputs than natural images due to the lack of shading or color information. Our approach is based on minimizing a joint energy function over input sketches, multi-view depth and surface normals, and point clouds. Our inference algorithm obtains a set of *depth maps* and *surface normals* of the shape from a collection of viewpoints using a *feed-forward network*. We then infer a dense *point cloud* that is consistent with the predicted depths and normals across all the viewpoints by minimizing our energy function. The point cloud is then converted to a discretized surface in the form of a polygon mesh and optionally further optimized to match the input line drawings more precisely.

Our approach appears to be the first that considers a learned, view-based representation for *generating* 3D shapes from sketches. The view-based representation allows us to process depth and normals at a considerably higher resolution and speed compared to voxel-based representations on existing hardware. Moreover, by incorporating the best of feed-forward architectures and mesh-based representations we are able to predict 3D shapes at a significantly higher quality. Finally, our architecture is trained on automatically generated, synthetic sketches of 3D shapes without requiring supervision in the form of human line drawings. Once trained, our method can generalize to reconstruct 3D shapes from human line drawings that can be approximate, noisy and not perfectly consistent across different viewing angles. Finally, as a by-product of our training procedure, our network also provides descriptors that can be used to perform sketch-based shape retrieval from 3D model collections. On two qualitatively different datasets (character models and man-made objects), our proposed approach achieves significantly better reconstruction results than alternative approaches in terms of several metrics (Hausdorff distance, Chamfer distance, voxel intersection over union, errors in depth and normal maps) and also based on a user study.

## 2. Related Work

**3D geometric inference from line drawings.** Compared to using natural images, estimating 3D shape from line drawings is considerably more challenging due to the lack of shading or texture information. Early works [60, 37, 36, 69] formulate the process of inferring a 3D shape based on reasoning about local geometric properties, such as convexity, parallelism, orthogonality and discontinuity, implied by lines and their intersections ("junctions"), to find a globally consistent shape. These approaches produce reasonable geometry when applied to specific families of polyhedral objects, but are less effective for organic shapes with smoothly varying surfaces. For smooth shapes, hand-designed rules are usually devised to extrude or elevate a 3D surface from contours [25, 42]. More recent methods enable the creation of freeform surfaces by exploiting geometric constraints present in specific types of line drawings, such as polyhedral scaffolds, cross-section lines and curvature flow lines [52, 65, 43]. All these methods derive geometric constraints from specific types of lines, require very accurate input drawings, and can only reconstruct what is drawn. On the other hand, various studies [32, 9] showed that humans can consistently interpret 3D shapes from sparse and approximate line drawings (up to a *bas-relief* transformation [3]). Although the exact mechanism of 3D shape perception in humans is not well understood, this indicates that pure geometric-based methods may not be able to mimic the human ability of shape understanding from sketches.

**Learning-based methods for shape synthesis.** In contrast to pure geometric methods, learning-based approaches argue that shape interpretation is fundamentally a learning problem, otherwise it is highly under-constrained. A large number of learning-based methods have focused on estimating 3D shapes from single, natural images that include color and texture. Early work was based on analyzing shading and texture cues within image regions [21, 51], while more recent work has employed ConvNets for predicting surface depth and normals from real images [12, 62]. Driven by the success of *encoder-decoder* architectures [33, 70, 27, 59, 26] that can effectively map inputs from one domain to another, newer methods use such architectures with convolutions in three dimensions to generate 3D shapes in a voxelized representation [63, 7, 67, 20, 47, 57]. A different line of work has employed ConvNets to model geometric transformations of an object to predict novel viewpoints [11, 55, 68, 71]. The approach of Tatarchenko *et al.* [56] is most related to ours. Their approach takes as input a single natural image and a viewpoint and uses a ConvNet to predict the color and depth from the provided viewpoint. They show compelling 3D reconstructions for chairs and cars from a single color image by projecting the depth maps from multiple views into a 3D space. Our approach is inspired by this work, but differs in a number of ways. Our method operates on line drawings, a more challenging type of input due to the lack of shading or color information. It predicts both normals and depth across multiple viewpoints, which are then integrated into a high-quality surface mesh representation through a joint optimization procedure. It also adapts a U-net architecture [26] along with multi-view decoder branches and a structured loss function to resolve ambiguities in the input line drawing. Finally, we provide a detailed comparison of view-based and voxel-based reconstruction approaches in terms of 3D shape evaluation metrics and a perceptual user study on various categories.

**Sketch-based 3D shape retrieval.** Sketch-based retrieval methods typically transform features of the input sketch and 3D shapes into a common space where comparisons can be made. Early work was based on hand-engineered descriptors [14, 45, 22, 34, 13, 66, 64, 53, 18], while more recently, ConvNets have been proposed to learn powerful representations for sketch-based retrieval [54, 61]. Unfortunately, these methods only allow retrieval of existing 3D shapes or parts. They provide no means to synthesize novel shapes or parts from scratch. A few recent approaches employ category-specific, predefined *parametric models* to guide shape reconstruction through ConvNets [40, 24, 19]. These methods are only able to recover specific shape parameters or rules from input sketches. If a drawing depicts a shape that cannot be described by the parameters of these models, then the reconstruction fails. In contrast, our method learns a representation capable of predicting shapes from sketches without any predefined parametric model. We expect 3D shape priors to automatically emerge in our deep network.

## 3. Method

Given a single, or multiple, hand-drawn sketches in the form of line drawings, our method aims to reconstruct a 3D shape. Line drawings are made by humans to convey shape information [10, 9]. They typically contain external contours (silhouettes) and internal contours to underlie salient shape features. We designed a deep network to automatically translate line drawings into 2D images representing surface depth and normals across several output viewpoints (Figure 1). The depth and normal predictions are then fused into a 3D point cloud, which is in turn converted into a polygon mesh. Although surface normals could be inferred by depth alone, we found that best reconstructions are achieved when both depth and normal predictions are made by the network and coherently fused into the point cloud.

Our network is trained to reconstruct multi-view depth and normal maps from either a single sketch depicting the shape from a particular input view (*e.g.*, front, side, or top), or from multiple sketches depicting the shape from different views (*e.g.*, front and side). A single sketch may not be sufficient to reconstruct the shape accurately, *e.g.*, the front side of an airplane does not explicitly convey information about its back. Hence, we consider the case where users provide multiple sketches as input at once, or provide them progressively while being guided by the intermediate shape reconstructions. In the latter case, users draw from one view, then our network, which is trained to reconstruct from that view, yields a 3D shape. Users can then draw a second sketch from another view, on top of the generated shape rendered semi-transparently from that view, similar to ShadowDraw [35] (see also our supplementary material for an example). Given the previous and new sketches as input, our network, trained to reconstruct from both views, yields an updated 3D shape. The process continues until users are satisfied with the result, at which point they may edit the mesh directly. In what follows, we discuss our network architecture

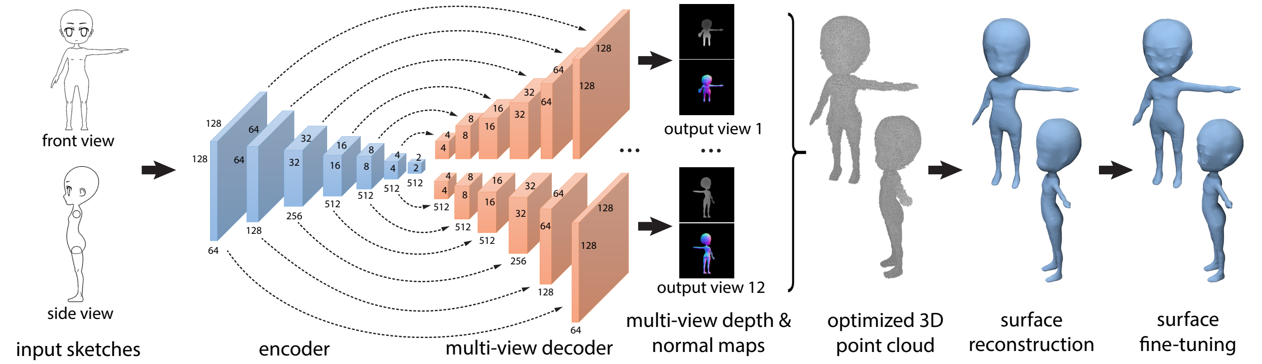

Figure 1. Our method takes line drawings as input and converts them into multi-view surface depth and normals maps from several output viewpoints via an encoder-multi-view-decoder architecture. The maps are fused into a coherent 3D point cloud, which is then converted into a surface mesh. Finally, the mesh can be further fine-tuned to match the input drawings more precisely through geometric deformations.

(Section 3.1) and training (Section 3.2). Then we discuss our optimization step to fuse the multi-view depth and normal maps into a single, coherent 3D point cloud and its conversion to a polygon mesh (Section 3.3).

### 3.1. Network Architecture

Our ConvNet takes as input line drawings from particular views of an object and outputs depth and normal maps in several, uniformly sampled output viewpoints (Figure 1). Our implementation uses 12 output viewpoints located at the equidistant vertices of a regular icosahedron. A camera is placed at each icosahedron vertex looking towards the center of the object and oriented towards the upright axis. All our training shapes are normalized such that they fit inside the icosahedron and are also consistently oriented.

**Input.** The input to our network are 256 × 256 intensity images representing the line drawings. When $C$ input sketches are available, they are concatenated as channels resulting in $256 \times 256 \times C$ dimensional input. For each input view configuration, we train a different network *i.e*., given a sketch representing the front of the object, we use the network trained to reconstruct the 3D shape from the front; or given two sketches representing the front and the top of the object, we use the network trained to reconstruct from the front and top (in this case, the two sketches are concatenated in this order). At first, this might seem restraining, yet we note that in many traditional CAD systems, it is common for users to use canonical views [48], and that better reconstruction results are achieved when the network is trained to reconstruct from specific rather than arbitrary views.

**Encoder.** The encoder network consists of a series of convolutional layers, all using kernel size of 4 and stride of 2. The filter size and number per layer is shown in Figure 1. All layers use batch normalization and leaky ReLUs (slope = 0.2) as activation functions. The output of the encoder is a 2 × 2 × 512 representation, which encodes shape information based on the input sketch(es). We note that this representation can be used for sketch-based shape retrieval.

**Decoder.** The decoder consists of 12 branches, each containing a series of upsampling and convolutional layers. The branches have the same layer structure but do not share parameters. Each branch takes as input the encoder's representation and outputs a 256 × 256 × 5 image for a corresponding output viewpoint. The 5-channel image includes a depth map (1 channel), a normal map (3 channels constrained to be unit norm) and a foreground probability map for that viewpoint. All pixels with probability more than 50% for foreground yield a binary mask indicating the projected surface area under that viewpoint. The output depth and normal maps are masked using this binary mask. Following the U-net architecture [49], the input to each convolutional layer is formed by the concatenation of the previous layer output in the decoder, and a corresponding layer output in the encoder (see Figure 1). The upsampling layers of the decoder upsample their input with a factor of 2. The convolutional layers use kernel size of 4 and stride of 1. Each convolutional layer is followed by batch normalization and leaky ReLU (slope = 0.2) as activation function. The first 3 layers in each decoder branch use dropout for regularization. The number and size of filters per layer in the decoder are shown in Figure 1. The output layer uses the tanh activation function since depths and normals lie in range $[-1, 1]$. Finally, the normal maps pass through an $\ell_2$ normalization layer that ensures they are unit length.

### 3.2. Training

To train our network, we need a dataset that includes 3D shapes along with corresponding training sketches. To create such dataset, one option would be to ask human subjects to provide us with line drawings depicting training 3D shapes. However, gathering human line drawings is labor-intensive and time-consuming. In contrast, we generated synthetic line drawings that approximate human line drawings based on well-known principles. Below we discuss the procedure we followed for sketch generation, then we discuss the objective used for training our network.

**Generating training sketches.** Non-photorealistic rendering algorithms can be used to create synthetic line drawings of 3D shapes. First, contours, or silhouettes, can be estimated by finding and connecting the set of points on the surface whose normal vector is perpendicular to the viewing direction [10]. Second, suggestive contours are extensions of contours that can be used to draw internal feature curves in shapes. These are found from zero-crossings of the radial curvature (surface curvature along viewing directions) [10]. Other types of internal feature curves include ridges and valleys, which are formed by the minima or maxima of the surface principal curvature values [41], or view-dependent curvature (in this case, the lines are called "apparent" ridges [28]). Another type of line drawings can be created through edge-preserving filtering [16] applied on images of shapes rendered under a simple shading scheme (e.g., Phong shading) [44]. All these feature curve definitions do not necessarily coincide each other [8]. We use a combination of these techniques to create several variants of line drawings per input shape. This also serves as a form of data augmentation. Specifically, for each shape and input view, we create 4 synthetic sketches by using: (i) silhouettes alone, (ii) silhouettes and suggestive contours, (iii) silhouettes, suggestive contours, ridges, valleys and apparent ridges, (iv) and edge-preserving filtering on rendered images of shapes. All training sketches and corresponding ground-truth depth and normal maps are rendered under orthographic projection according to our output viewpoint setting. Using perspective projection could also be an option, however, since depth has a relatively short range for our rendered objects, the differences in the resulting images tend to be small.

**Loss function.** Given training sketches of shapes along with the corresponding foreground, depth and normal maps for our output viewpoints, we attempt to estimate the network parameters to minimize a loss function. Our loss function consists of four terms penalizing (a) differences between the training depth maps and predicted depth maps, (b) angle differences between the training normal maps and predicted normal maps, (c) disagreement between ground-truth and predicted foreground masks, (d) large-scale structural differences between the predicted maps and the training maps. Specifically, given $T$ training sketches along with ground-truth foreground, depth and normal maps for our $V$ output viewpoints, our loss function is a combination of the following terms described in the following paragraphs:

$$L=\sum_{t=1}^{T}(\lambda_1 L_{depth}(t)+\lambda_2 L_{normal}(t)+\lambda_3 L_{mask}(t)+\lambda_4 L_{adv}(t))$$

where $\lambda_1 = 1.0, \lambda_2 = 1.0, \lambda_3 = 1.0, \lambda_4 = 0.01$ are weights tuned in a hold-out validation set.

**Per-pixel depth and normal loss.** The first two terms consider per-pixel differences in the predicted depths and normals with respect to ground-truth. Specifically, we use $\ell_1$ distance for depths and angle cosine differences for normal directions. The depth and normal differences are computed only for pixels marked as foreground in the ground-truth:

$$L_{depth}(t) = \sum_{p,v} \left( |d_{p,v}(\mathbf{S}_t) - \hat{d}_{p,v,t}| \right) \hat{f}_{p,v,t}$$

$$L_{normal}(t) = \sum_{p,v} \left(1 - \mathbf{n}_{p,v}(\mathbf{S}_t) \cdot \hat{\mathbf{n}}_{p,v,t}\right) \hat{f}_{p,v,t}$$

where $\mathbf{S}_t$ is a training sketch, $\hat{d}_{p,v,t}$ and $\hat{\mathbf{n}}_{p,v,t}$ are ground-truth depth and normal for the pixel $p$ in viewpoint $v$. Each pixel has a ground-truth binary label $\hat{f}_{p,v,t}$, which is 1 for foreground, and 0 otherwise. The depth and normal predictions for the sketch $\mathbf{S}_t$ are denoted as $d_{p,v}(\mathbf{S}_t)$ and $\mathbf{n}_{p,v}(\mathbf{S}_t)$ respectively. We note that all training depths are normalized within the range $[-1, 1]$ while predicted depths are also clamped in this range. Thus both terms above have comparable scale (*i.e.*, both range between $[0, 2]$ per pixel). We also note that we tried $\ell_2$ distance for penalizing depth differences but this tended to produce less sharp maps.

**Mask loss.** Penalizing disagreement between predicted and ground-truth foreground labeling can be performed via the cross-entropy function commonly used in classification.

**Adversarial loss.** We also penalize structural differences in the output maps with respect to ground-truth through an "adversarial" network. This has been shown to serve as an effective prior for various image-to-image transformation tasks [26]. The adversarial loss term takes as input a 5-channel image $\mathbf{I}$ that concatenates the depth channel, the 3 normal channels, and foreground map channel produced by the decoder per viewpoint, and outputs the probability for these maps to be "real": $\mathcal{L}_{\text{adv}} = -\sum_v \log P(\text{“real”}|\mathbf{I})$. The probability is estimated using the "adversarial" network trained to discriminate ground-truth ("real") maps $\hat{\mathbf{I}}$ from generated ("fake") maps $\mathbf{I}$. Both networks are trained alternatively using the technique of [17]. The adversarial network architecture is the same as the encoder except the last layer that maps the output to probabilities via a fully-connected layer followed by a sigmoid activation.

## 3.3. Point Cloud and Mesh Generation

Given multi-view depth and normal maps produced by our network at test time, our next goal is to consolidate them into a single, coherent 3D point cloud. The depth and normal predictions produced by the network are not guaranteed to be perfect or even consistent i.e., the derivatives of the predicted depth might not entirely agree with the predicted normals, or the predicted depths for common surface regions across different viewpoints might not yield exactly the same 3D points. Below we discuss an optimization approach to fuse all multi-view depth and normal map predictions into a coherent 3D point cloud, then we discuss mesh generation and post-processing to match the input sketches more precisely. Our optimization approach shares similarities with bundle adjustment and multi-view reconstruction [58, 15]. In our case, our output viewpoints are fixed and we use the normal maps in our energy minimization to promote consistency between depth derivatives and surface normals.

**Multi-view depth and normal map fusion.** The first step of the fusion process is to map all foreground pixels to 3D points. Each pixel is considered foreground if its predicted probability in the foreground map is above 50%. Given the depth $d_{p,v}$ of a foreground pixel $p$ with image-space coordinates $\{p_x, p_y\}$ in the output map of a viewpoint $v$, a 3D point $\mathbf{q}_{p,v}$ can be generated according to the known extrinsic camera parameters (coordinate frame rotation $\mathbf{R}_v$ and translation $\mathbf{e}_v$ in object space). Under the assumed orthographic projection, the point position is computed as:

$$\mathbf{q}_{p,v} = \mathbf{R}_v \left[\kappa p_x \quad \kappa p_y \quad d_{p,v}\ \right]^T + \mathbf{e}_v$$

where $\kappa$ is a known scaling factor, representing the distance between two adjacent pixel centers when their centers are mapped to object space. Each point is also equipped with a normal $\mathbf{n}_{p,v}$ based on the predicted normal map. The result of this first step is a generated point set per view. In a second step, we run ICP [50] to rigidly align all-pairs of point sets, which helps dealing with inconsistencies in the predicted depth maps.

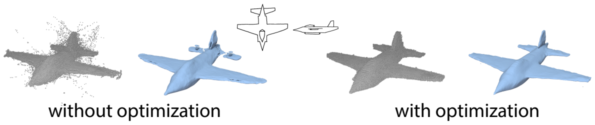

Figure 2. Without optimization the noisy point cloud will lead to misaligned regions in the reconstructed shape.

A naive reconstruction method would be to simply concatenate all aligned point sets from all output views into a single point cloud. However, such approach often results in a noisy point cloud with misaligned regions due to the remaining depth map inconsistencies not handled by ICP. The effect of these inconsistencies tends to be amplified during mesh generation, since a smooth surface cannot pass through all the misaligned regions (Figure 2). Our optimization procedure aims to deal with this problem. Specifically, we treat the depths of all pixels as variables we want to optimize for. The pixel depths are optimized such that (a) they are close to the predicted (approximate) depths produced by the network, (b) their first-order derivatives yield surface tangent vectors that are as-orthogonal-as-possible to the predicted normals, (c) they are consistent with depths and normals of corresponding 3D points generated in other viewpoints. These requirements are expressed in a single energy over all pixel depths $\mathbf{D} = \{d_{p,v}\}$ with terms imposing the above three conditions, as explained in the next paragraphs:

$$E(\mathbf{D}) = E_{net}(\mathbf{D}) + E_{orth}(\mathbf{D}) + E_{cons}(\mathbf{D})$$

**Network prediction term.** The term $E_{net}(\mathbf{D})$ penalizes deviation from the approximate depths $\tilde{d}_{p,v}(\mathbf{S}_t)$ produced from the network at each pixel $p$ and viewpoint $v$:

$$E_{net}(\mathbf{D}) = w_1 \sum_{p,v} (d_{p,v} - \tilde{d}_{p,v}(\mathbf{S}_t))^2$$

where $w_1$ weights this term (set to 1.0 through hold-out validation). We use $\ell_2$ norm here so that the energy minimization yields a linear system that can be solved efficiently.

**Orthogonality term.** The term $E_{orth}(\mathbf{D})$ penalizes deviation from orthogonality between surface tangents, approximated by first-order depth derivatives, and predicted surface normals $\tilde{\mathbf{n}}_{p,v}(\mathbf{S}_t)$. Given a 3D point $\mathbf{q}_{p,v}$ generated for pixel $p$ and viewpoint $v$, we estimate two surface tangent directions based on first-order depth derivatives [39]:

$$\mathbf{t}^{(x)}_{p,v} = \begin{bmatrix} \kappa & 0 & \dfrac{\partial d_{p,v}}{\partial x} \end{bmatrix}^T, \quad \mathbf{t}^{(y)}_{p,v} = \begin{bmatrix} 0 & \kappa & \dfrac{\partial d_{p,v}}{\partial y} \end{bmatrix}^T$$

The derivatives can be approximated with a horizontal and vertical gradient filter that is convolved with depths in a $3 \times 3$ neighborhood around $p$. The energy term is expressed as:

$$E_{orth}(\mathbf{D}) = w_2 \sum_{p,v} [(\mathbf{t}^{(x)}_{p,v} \cdot \tilde{\mathbf{n}}_{p,v}(\mathbf{S}_t))^2 + (\mathbf{t}^{(y)}_{p,v} \cdot \tilde{\mathbf{n}}_{p,v}(\mathbf{S}_t))^2]$$

where $w_2$ is a weight (set to 1.0 through holdout validation). Since the derivatives are unreliable near the shape silhouette, we omit silhouette points for each view from this term.

**View consistency term.** Given a 3D point $\mathbf{q}_{p,v}$ generated from pixel $p$ at viewpoint $v$, we can calculate its depth with respect to the image plane of another viewpoint $v'$ as well as the pixel that it is projected onto as: $p' = \Pi_{v'}(\mathbf{q}_{p,v})$, where $\Pi_{v'}$ denotes orthographic projection based on the parameters of viewpoint $v'$. When the 3D point is not occluded and falls within the image formed at viewpoint $v'$, the calculated depth $d_{v'}(\mathbf{q}_{p,v})$ of that point should be in agreement with the depth $d_{p',v'}$ stored in the corresponding pixel $p'$ of the viewpoint $v'$. Similarly, the normal of that point $\mathbf{n}_{v'}(\mathbf{q}_{p,v})$ relative to the viewpoint $v'$ should be as-orthogonal-as possible to the surface tangent vector, approximated by the derivative of the depth stored in the corresponding pixel $p'$. The view consistency term penalizes: (a) squared differences between the depth at each pixel and the calculated depth of all 3D points projected onto that pixel, (b) deviation from orthogonality between the surface tangent vector at each pixel and the normal of all 3D points projected onto that pixel. The term is expressed as follows:

$$E_{cons}(\mathbf{D}) = w_3 \sum_{\substack{p,v,p',v': \\ p'=\Pi_{v'}(\mathbf{q}_{p,v})}} (d_{p',v'} - d_{v'}(\mathbf{q}_{p,v}))^2 +$$
$$+ w_4 \sum_{\substack{p,v,p',v': \\ p'=\Pi_{v'}(\mathbf{q}_{p,v})}} (\mathbf{t}^{(x)}_{p',v'} \cdot \mathbf{n}_{v'}(\mathbf{q}_{p,v}))^2 + (\mathbf{t}^{(y)}_{p',v'} \cdot \mathbf{n}_{v'}(\mathbf{q}_{p,v}))^2$$

where $w_3$ and $w_4$ are weights both set to 0.3. We note that if a 3D point is projected onto a pixel that is masked as background (thus, its depth is invalid), then we exclude that pixel from the above summation. If the 3D point is projected onto background pixels in the majority of views, then this means that the point is likely an outlier and we remove it from the point cloud. As a result, there are few $(p, p')$ pixel pairs in the above equation: each foreground pixel often has 3-4 corresponding pixels in other views.

**Energy minimization.** The energy is quadratic in the unknown pixel depths, thus we can minimize it by solving a linear system. Due to the orthogonality term, which involves a linear combination (filtering) of depths within a pixel neighborhood, the depth of each pixel cannot be solved independently of the rest of the pixels. The solution can be computed through a sparse linear system - we provide its solution in our supplementary material. When we compute the pixel depths, the corresponding 3D point positions, generated by these pixels, are updated. Given new 3D point positions, the consistency term also needs updating since the points might now be projected onto different pixels. This gives rise to an iterative scheme, where at each step we estimate pixel depths by solving the linear system, then update the 3D point positions. We observed that the depth estimates become increasingly consistent across different views at each iteration and practically convergence is achieved after 3-5 iterations. As shown in Figure 2, the resulting point cloud yields a smoother reconstructed surface.

**Mesh reconstruction and fine-tuning.** We apply the screened Poisson Surface Reconstruction algorithm [29] to convert the resulting point cloud and normals to a surface mesh. Our method can optionally further "fine-tune" the generated mesh so that it matches the input contours more precisely. To do this, for each input line drawing we first extract its external contours and discretize them into a dense set of 2D points. Then for each input view, we render the mesh under the same orthographic projection, and find nearest corresponding mesh points to each contour point under this projection. Then we smoothly deform the 3D mesh such that the projected mesh points move towards the contour points under the constraint that the surface Laplacians [38], capturing underlying surface details, are preserved. We also deform the mesh so that it better matches the internal contours of the sketch. This is done by finding nearest corresponding mesh points to each internal contour point and scaling their Laplacian according to the scheme proposed in [38]. Mesh deformation is executed by solving a sparse linear system involving all constraints from all internal and external contours across all input views. Figure 1 shows a reconstructed mesh before and after fine-tuning.

**Implementation.** The network is implemented in Tensorflow [1]. Training takes about 2 days for 10K training meshes (40K training sketches) on a TitanX GPU. We use the Adam solver [30] (hyperparameters $\beta 1$ and $\beta 2$ are set to 0.9 and 0.999 respectively). At test time, processing input sketches through the network takes 1.5 sec on a TitanX GPU, fusing the depth and normal maps takes 3 sec, mesh reconstruction and fine-tuning takes 4 sec (fusion and mesh reconstruction are implemented on the CPU - running times are reported on a dual Xeon E5-2699v3). In total, our method takes about 10 seconds to output a shape. Our source code and datasets are available on our project page: https://people.cs.umass.edu/~zlun/SketchModeling

## 4. Evaluation

We now discuss the experimental evaluation of our method.

**Datasets.** To train our network, we gathered three collections of 3D shapes along with their synthetic sketches. Each of the collections included shapes belonging to the same broad category. The categories were 3D computer characters, airplanes, and chairs. To create the 3D computer character collection, we downloaded freely available 3D models of characters from an online repository ("The Models Resource" [46]). The collection contained humanoid, alien, and other fictional 3D models of characters. The airplanes and chairs originated from 3D ShapeNet [6]. We used these particular categories from ShapeNet because the shapes in these categories have large geometric and structural variation. Table 1 reports the number of training shapes and view setting used to generate the training sketches.

| | #training shapes | view A | view B |
|---|---|---|---|
| Character | 10000 | front | side |
| Airplane | 3667 | top | side |
| Chair | 9573 | front | side |

Table 1. Training dataset statistics.

**Test dataset.** To evaluate our method and compare it with alternatives, we created a test dataset of synthetic and human line drawings for each of the above categories. Each line drawing was created according to a reference test shape. The goal of the evaluation was to examine how well the reconstructed 3D shapes from these test line drawings matched the reference test shapes. To execute a proper evaluation, the reference test shapes should be sufficiently different from all training shapes. Otherwise, by overfitting a network to the training dataset or by simply using a nearest neighbor sketch-based retrieval approach, one could perfectly reproduce the reference shapes. To create the test dataset of reference shapes, one option would be to randomly split the above collections into a training and test part. However, a problem with this strategy is that several test shapes would be overly similar to one or more training shapes because of duplicate, or near-duplicate, 3D models that often exist in these collections (i.e., models that are identical up to an affine transformation, having tiny part differences or different mesh resolution). To create our test dataset, we found 120 shapes (40 per category) in our collections that we ensured to be sufficiently different from the shapes used for training by performing two checks. First, for each shape, we aligned it to each other shape in the collection through the best matching affine transformation and compute their Chamfer distance. The Chamfer distance is computed by measuring the distance of each of the points on one shape to the nearest surface point on the other shape, then the average of these distances is used (we sampled 10K points uniformly per shape). We verified that the Chamfer distance between each test shape and its nearest training shape is well above a threshold. Second, we rendered synthetic sketches for each shape based on the input views per category and extracted the representation from our encoder for these sketches. We then retrieved the nearest other shape based on Euclidean distance over the sketch representations. We verified that the distance is well above a threshold. We also visually confirmed that test and training shapes were different and the selected thresholds were appropriate.

For our 120 test shapes, we produced synthetic sketches for 90 of them (30 per category), and gathered human line drawings for the remaining 30 shapes (10 per category). Synthetic sketches were produced from the test shapes using the line rendering techniques described in Section 3.2 based on the input views A and B per category (Table 1). The human sketches were produced by asking two artists to provide us with hand-drawn line drawings of reference test shapes. The test shapes were presented to the artists on a computer display and were rendered using Phong shading. Their views were selected to approximately match the input views A and B per category. We asked the artists to create on paper line drawings depicting the presented shapes based on the selected views. We then scanned their line drawings, cropped and scaled them so that the scanned drawn area

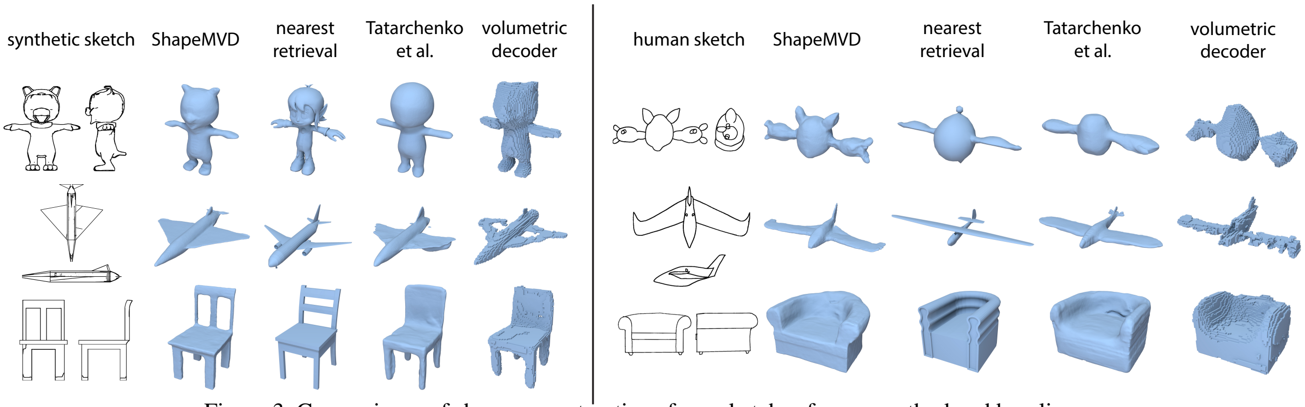

Figure 3. Comparisons of shape reconstructions from sketches for our method and baselines.

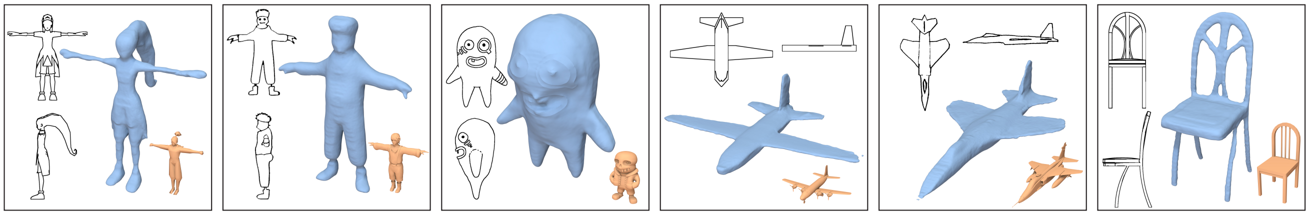

Figure 4. Gallery of results. Blue shapes represent reconstructions produced by our method from the input sketches. Orange shapes are the nearest shapes in the training datasets retrieved via sketch-based retrieval.

matches the drawing area of training sketches on average. In contrast to synthetic sketches, human line drawings tend to be noisy and inconsistent across different views.

**Evaluation measures.** Given the above test sketches as input, the goal of our evaluation is to measure how well the 3D shapes reconstructed by various methods, including ours, matched the reference test shapes used to produce these sketches. Our method and the alternatives, listed in the following paragraphs, were trained and tested separately on each category using the same splits. We used five evaluation measures to compare the reconstructed shapes to the reference ones: Chamfer distance, Hausdorff distance, surface normal distance, depth map error, volumetric Jaccard distance. The Hausdorff distance is computed by measuring the distance of each surface point on the reconstructed shape to the nearest surface point on the reference shape, then computing the maximum of these distances. The surface normal distance is computed by measuring the angle between the normal at each surface point on the reconstructed shape and the normal at the nearest surface point on the reference shape, then computing the mean of the angles. The depth map error is computed by measuring the absolute differences between pixel depths in each of the output depth maps produced by our network and the corresponding depth maps of the reference shape, then computing the average depth differences. To compute the volumetric Jaccard distance, we voxelized the reconstructed and reference shapes in a $128\times128\times128$ binary grid and measured the number of voxels commonly filled in both shapes (their volume intersection) divided by the number of their filled voxels (union of their volumes) - this is the Intersection over Union ($IoU$). We use $1 - IoU$ as the volumetric Jaccard distance.

**Comparisons.** We tested the reconstructions produced by our method (called "ShapeMVD") versus the following methods: (a) a network based on the same encoder as ours but using a volumetric decoder baseline instead of our multi-view decoder, (b) a network based on the same encoder as ours but with the Tatarchenko *et al.*'s view-based decoder [56] instead of our multi-view decoder, (c) the convolutional 3D LSTM architecture (R2N2) provided by Choy *et al.* 's implementation [7], and (d) nearest sketch-based shape retrieval. For the volumetric decoder baseline (a), we used a $128 \times 128 \times 128$ output binary grid (the maximum we could fit in 12GB GPU memory). To make sure that the comparison is fair, we set the number of parameters in the volumetric decoder such that it is comparable to the number of parameters in our decoder. The volumetric decoder consisted of five transpose 3D convolutions of stride 2 and kernel size $4 \times 4 \times 4$. The number of filters starts with 512 and is divided by 2 at each layer. Leaky ReLU functions and batch normalization were used after each layer. We note that we did not use skip-connections (U-net architecture) in the volumetric decoder because the size of the feature representations produced in the sketch image-based encoder is incompatible with the ones produced in the decoder. For Tatarchenko *et al.*'s method, the viewpoint is encoded into a continuous $64\times1$ representation passed as input to the view-based decoder described in [56] without separate branches. To ensure a fair comparison, we increased the number of filters per up-convolutional layer by a factor of 3 so that the number of parameters in their and our decoder is comparable. We also train it with the same loss function as ours. We additionally implemented a variant of Tatarchenko *et al.*'s decoder by adding U-net connections between the encoder and their decoder. We report the evaluation measures on this additional variation. For the nearest-neighbor baseline, we extract the representation of the input test sketches based on our encoder. This is used as a query representation

| | Man-made objects (synthetic) | | | | | | Character models (synthetic) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ShapeMVD | nearest retrieval | Tatarchenko et al.[56] | [56]+ U-net | volumetric decoder | R2N2 [7] | ShapeMVD | nearest retrieval | Tatarchenko et al.[56] | [56]+ U-net | volumetric decoder | R2N2 [7] |
| Hausdorff distance | **0.092** | 0.165 | 0.142 | 0.121 | 0.113 | 0.144 | **0.089** | 0.200 | 0.119 | 0.092 | 0.152 | 0.148 |
| Chamfer distance | **0.015** | 0.025 | 0.022 | 0.017 | 0.021 | 0.026 | **0.015** | 0.036 | 0.025 | 0.016 | 0.026 | 0.032 |
| normal distance | **30.66** | 42.57 | 35.58 | 32.32 | 49.40 | 48.78 | **30.61** | 44.93 | 34.98 | 31.00 | 53.84 | 53.13 |
| depth map error | **0.026** | 0.049 | 0.039 | 0.030 | 0.038 | 0.045 | **0.018** | 0.040 | 0.030 | 0.019 | 0.031 | 0.036 |
| volumetric distance | **0.344** | 0.501 | 0.442 | 0.374 | 0.432 | 0.512 | **0.313** | 0.541 | 0.428 | 0.329 | 0.437 | 0.493 |
| | Man-made objects (human drawing) | | | | | | Character models (human drawing) | | | | | |
| | ShapeMVD | nearest retrieval | Tatarchenko et al.[56] | [56]+ U-net | volumetric decoder | R2N2 [7] | ShapeMVD | nearest retrieval | Tatarchenko et al.[56] | [56]+ U-net | volumetric decoder | R2N2 [7] |
| Hausdorff distance | **0.116** | 0.176 | 0.153 | 0.153 | 0.130 | 0.149 | **0.117** | 0.188 | 0.139 | 0.136 | 0.178 | 0.168 |
| Chamfer distance | **0.017** | 0.031 | 0.024 | 0.025 | 0.022 | 0.028 | **0.021** | 0.036 | 0.025 | 0.024 | 0.032 | 0.036 |
| normal distance | **27.04** | 40.96 | 32.40 | 30.45 | 48.32 | 48.12 | **33.44** | 43.81 | 36.11 | 34.74 | 54.91 | 54.29 |
| depth map error | **0.021** | 0.042 | 0.033 | 0.032 | 0.032 | 0.042 | **0.026** | 0.040 | 0.031 | 0.027 | 0.037 | 0.040 |
| volumetric distance | **0.311** | 0.544 | 0.405 | 0.403 | 0.405 | 0.500 | **0.298** | 0.458 | 0.342 | 0.307 | 0.420 | 0.436 |

Table 2. Comparisons of our method with baselines based on our evaluation measures (the lower the numbers, the better)

to retrieve the training shape whose sketches have the nearest encoder representation based on Euclidean distance. All methods had access to the same training dataset per category and were evaluated on the same test set.

Table 2 reports the evaluation measures for all competing methods based on both synthetic and human line drawings. We include evaluation separately for organic shapes (3D character collection) and man-made shapes (measures are averaged over airplanes and chairs). We also include standard deviations in our supplementary material. Our method produces much more accurate reconstructions than the competing methods in all cases. We note that mesh fine-tuning was not used here for any of the methods. The reason was to evaluate the methods by factoring out the post-processing effects of fine-tuning. Fine-tuning is optional and does not significantly affect the errors. It is used only to add details ("stylize") the produced meshes based on the input contours when these are precisely drawn, and if users desire so (we provide more discussion regarding the effects of fine-tuning on the evaluation measures in the supplementary material). With respect to Tatarchenko *et al.*'s method, we find that its enhancement with U-net connections improves its performance, but still performs worse than our method, especially for man-made objects. This implies that U-net is a significant enhancement. We finally observe that the R2N2 does not perform better than our volumetric decoder baseline. Figure 3 shows representative input test sketches, and output meshes for competing methods (again, no fine-tuning is used here). In general, the nearest neighbor results look plausible because retrieval returns human-modeled training shapes with fine details (e.g., facial features). Such details are not captured by any of the methods, including ours. On the other hand, as shown in the figure, and confirmed by numerical evaluation, compared to nearest neighbor retrieval and other methods, ours produces shapes that better match the input sketch. The main reason is that our method better preserves the shape structure, topology and coarse geometry depicted in the input sketch. From this aspect, we believe that the shapes reconstructed by our method may serve as better starting "proxies" for artists to further improve upon. We also conducted an Amazon Mechanical Turk user study to perceptually evaluate the results of the methods. Specifically, we asked human subjects to compare the produced shapes from different methods and select the one that best matches the input sketch. Our method was chosen by human participants to be the one producing shapes that best match the input sketches most of the time compared to the other methods including nearest retrieval (see supplementary material for results and more details on our user study).

**More results.** The supplementary material includes all reconstructed test shapes for our method, nearest neighbors and competing methods (fine-tuning is not used on any of these results). We also include evaluation of our method against degraded variants (e.g., using depth only, skipping the fusion step or GAN), and results using two sketches versus one sketch as input. Figure 4 shows shapes produced by our method for various input synthetic and human sketches. Fine-tuning was used for the meshes of this figure.

## 5. Conclusion

We presented an approach for 3D shape reconstruction from sketches. Our method employs a ConvNet to predict depth and normals from a set of viewpoints, and the resulting information is consolidated into a 3D point cloud via energy minimization. We evaluated our method and variants on two qualitatively different categories (characters and man-made objects). Our results indicate that view-based reconstruction of a 3D shape is significantly more accurate than voxel-based reconstruction. We also showed that our method can generalize to human-drawn sketches. We believe that there is significant room for improving our method in the future. For example, it would be interesting to explore the possibility of incorporating the fusion process in the network, and modifying its architecture such that reconstruction is done from arbitrary viewpoints. Our reconstructed shapes often lack fine details that users would prefer to see in production-quality 3D models. We believe that these shapes can serve as starting "proxies" for artists to improve upon through modeling interfaces. From this aspect, it would be useful to integrate interactive modeling techniques into our method.

**Acknowledgements.** We acknowledge support from NSF (CHS-1422441,CHS-1617333,IIS-1617917,IIS-1423082), Adobe, NVidia, Facebook. We acknowledge the MassTech Collaborative grant for funding the UMass GPU cluster.

## References

[1] TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 6

[2] B. Allen, B. Curless, and Z. Popović. The space of human body shapes: reconstruction and parameterization from range scans. In *Proc. SIGGRAPH*, 2003. 1

[3] P. N. Belhumeur, D. J. Kriegman, and A. L. Yuille. The bas-relief ambiguity. *International journal of computer vision*, 35(1), 1999. 2

[4] V. Blanz and T. Vetter. A morphable model for the synthesis of 3d faces. In *Proc. SIGGRAPH*, 1999. 1

[5] T. J. Cashman and A. W. Fitzgibbon. What shape are dolphins? building 3d morphable models from 2d images. *IEEE transactions on pattern analysis and machine intelligence*, 35(1), 2013. 1

[6] A. X. Chang, T. A. Funkhouser, L. J. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. Shapenet: An information-rich 3d model repository. In *arXiv, abs/1512.03012*. 2015. 6

[7] 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 *Proc. ECCV*, 2016. 1, 2, 7, 8

[8] F. Cole, A. Golovinskiy, A. Limpaecher, H. S. Barros, A. Finkelstein, T. Funkhouser, and S. Rusinkiewicz. Where do people draw lines? *ACM Trans. Graph.*, 27(3), 2008. 4

[9] F. Cole, K. Sanik, D. DeCarlo, A. Finkelstein, T. Funkhouser, S. Rusinkiewicz, and M. Singh. How well do line drawings depict shape? *ACM Trans. Graph.*, 28(3), 2009. 2

[10] D. DeCarlo, A. Finkelstein, S. Rusinkiewicz, and A. Santella. Suggestive contours for conveying shape. *ACM Trans. Graph.*, 22(3), 2003. 2, 3, 4

[11] A. Dosovitskiy, J. Tobias Springenberg, and T. Brox. Learning to generate chairs with convolutional neural networks. In *Proc. CVPR*, 2015. 1, 2

[12] D. Eigen and R. Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proc. ICCV*, 2015. 2

[13] M. Eitz, R. Richter, T. Boubekeur, K. Hildebrand, and M. Alexa. Sketch-based shape retrieval. *ACM Trans. Graph.*, 31(4), 2012. 2

[14] T. Funkhouser, P. Min, M. Kazhdan, J. Chen, A. Halderman, D. Dobkin, and D. Jacobs. A search engine for 3d models. *ACM Trans. Graph.*, 22(1), 2003. 2

[15] S. Galliani and K. Schindler. Just look at the image: viewpoint-specific surface normal prediction for improved multi-view reconstruction. In *Proc. CVPR*, 2016. 4

[16] E. S. L. Gastal and M. M. Oliveira. Domain transform for edge-aware image and video processing. *ACM Trans. Graph.*, 30(4), 2011. 4

[17] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Proc. NIPS*, 2014. 4

[18] X. Guo, J. Lin, K. Xu, S. Chaudhuri, and X. Jin. CustomCut: On-demand Extraction of Customized 3D Parts with 2D Sketches. *Computer Graphics Forum*, 2016. 2

[19] X. Han, C. Gao, and Y. Yu. Deepsketch2face: A deep learning based sketching system for 3d face and caricature modeling. *ACM Transactions on Graphics*, 36(4), 2017. 2

[20] C. Häne, S. Tulsiani, and J. Malik. Hierarchical surface prediction for 3d object reconstruction. In *arXiv, abs/1704.00710*. 2017. 2

[21] D. Hoiem, A. A. Efros, and M. Hebert. Geometric context from a single image. In *Proc. ICCV*, 2005. 2

[22] S. Hou and K. Ramani. Sketch-based 3d engineering part class browsing and retrieval. In *Proc. SBIM*, 2006. 2

[23] H. Huang, E. Kalogerakis, and B. Marlin. Analysis and synthesis of 3d shape families via deep-learned generative models of surfaces. *Computer Graphics Forum*, 34(5), 2015. 1

[24] H. Huang, E. Kalogerakis, E. Yumer, and R. Mech. Shape synthesis from sketches via procedural models and convolutional networks. *IEEE Transactions Visualization and Computer Graphics*, 2017. 2

[25] T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy: A sketching interface for 3d freeform design. In *Proc. SIGGRAPH*, 1999. 2

[26] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *Proc. CVPR*, 2017. 1, 2, 4

[27] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *Proc. ECCV*, 2016. 1, 2

[28] T. Judd, F. Durand, and E. H. Adelson. Apparent ridges for line drawing. *ACM Trans. Graph.*, 26(3), 2007. 4

[29] M. Kazhdan and H. Hoppe. Screened poisson surface reconstruction. *ACM Trans. Graph.*, 32(3), 2013. 6

[30] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *arXiv, abs/1412.6980*. 2014. 6

[31] J. J. Koenderink. What does the occluding contour tell us about solid shape? *Perception*, 13(3), 1984. 1

[32] J. J. Koenderink, A. J. Van Doorn, and A. M. Kappers. Surface perception in pictures. *Attention, Perception, & Psychophysics*, 52(5), 1992. 2

[33] G. Larsson, M. Maire, and G. Shakhnarovich. Learning representations for automatic colorization. In *Proc. ECCV*, 2016. 1, 2

[34] J. Lee and T. Funkhouser. Sketch-based search and composition of 3d models. In *Proc. SBM*, 2008. 2

[35] Y. J. Lee, C. L. Zitnick, and M. F. Cohen. Shadowdraw: Real-time user guidance for freehand drawing. *ACM Trans. Graph.*, 30(4), 2011. 2

[36] H. Lipson and M. Shpitalni. Optimization-based reconstruction of a 3d object from a single freehand line drawing. *Computer-Aided Design*, 28, 1996. 1

[37] J. Malik. Interpreting line drawings of curved objects. *International Journal of Computer Vision*, 1(1), 1987. 1

[38] A. Nealen, O. Sorkine, M. Alexa, and D. Cohen-Or. A sketch-based interface for detail-preserving mesh editing. *ACM Trans. Graph.*, 24(3), 2005. 6

[39] D. Nehab, S. Rusinkiewicz, J. Davis, and R. Ramamoorthi. Efficiently combining positions and normals for precise 3d geometry. *ACM Trans. Graph.*, 24(3), 2005. 5

[40] G. Nishida, I. Garcia-Dorado, D. G. Aliaga, B. Benes, and A. Bousseau. Interactive sketching of urban procedural models. *ACM Trans. Graph.*, 2016. 2

[41] Y. Ohtake, A. Belyaev, and H.-P. Seidel. Ridge-valley lines on meshes via implicit surface fitting. *ACM Trans. Graph.*, 23(3), 2004. 4

[42] L. Olsen, F. F. Samavati, M. C. Sousa, and J. A. Jorge. Sketch-based modeling: A survey. *Computers & Graphics*, 33(1), 2009. 2

[43] H. Pan, Y. Liu, A. Sheffer, N. Vining, C.-J. Li, and W. Wang. Flow aligned surfacing of curve networks. *ACM Trans. Graph.*, 34(4), 2015. 2

[44] B. T. Phong. Illumination for computer generated pictures. *Commun. ACM*, 18(6), 1975. 4

[45] J. Pu, K. Lou, and K. Ramani. A 2d sketch-based user interface for 3d cad model retrieval. *Computer-Aided Design and Applications*, 2(6), 2005. 2

[46] T. M. Resource. https://www.models-resource.com/, 2017. 6

[47] G. Riegler, A. O. Ulusoy, H. Bischof, and A. Geiger. Octnetfusion: Learning depth fusion from data. In *arXiv, abs/1704.01047*. 2017. 2

[48] A. Rivers, F. Durand, and T. Igarashi. 3d modeling with silhouettes. *ACM Trans. Graph.*, 29(4), 2010. 3

[49] O. Ronneberger, P.Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Proc. Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351, 2015. 3

[50] S. Rusinkiewicz and M. Levoy. Efficient variants of the icp algorithm. In *Proc. 3D Digital Imaging and Modeling*, 2001. 5

[51] A. Saxena, M. Sun, and A. Y. Ng. Make3d: Learning 3d scene structure from a single still image. *IEEE transactions on pattern analysis and machine intelligence*, 31(5), 2009. 2

[52] R. Schmidt, A. Khan, K. Singh, and G. Kurtenbach. Analytic drawing of 3d scaffolds. *ACM Trans. Graph.*, 28(5), 2009. 2

[53] R. G. Schneider and T. Tuytelaars. Sketch classification and classification-driven analysis using fisher vectors. *ACM Trans. Graph.*, 33(6), 2014. 2

[54] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller. Multiview convolutional neural networks for 3d shape recognition. In *Proc. ICCV*, 2015. 2

[55] M. Tatarchenko, A. Dosovitskiy, and T. Brox. Single-view to multi-view: Reconstructing unseen views with a convolutional network. In *arXiv, abs/1511.06702*. 2015. 1, 2

[56] M. Tatarchenko, A. Dosovitskiy, and T. Brox. Multi-view 3d models from single images with a convolutional network. In *Proc. ECCV*, 2016. 1, 2, 7, 8

[57] M. Tatarchenko, A. Dosovitskiy, and T. Brox. Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. In *arXiv, abs/1703.09438*. 2017. 2

[58] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon. Bundle adjustment - a modern synthesis. In *Proc. ICCV Workshop on Vision Algorithms: Theory and Practice*, 2000. 4

[59] D. Ulyanov, V. Lebedev, A. Vedaldi, and V. Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. In *Proc. ICML*, 2016. 1, 2

[60] D. Waltz. Understanding line drawings of scenes with shadows." the psychology of computer vision. patrick henry winston, ed, 1975. 1

[61] F. Wang, L. Kang, and Y. Li. Sketch-based 3d shape retrieval using convolutional neural networks. In *Proc. CVPR*, 2015. 2

[62] X. Wang, D. Fouhey, and A. Gupta. Designing deep networks for surface normal estimation. In *Proc. CVPR*, 2015. 2

[63] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Proc. NIPS*, 2016. 1, 2

[64] X. Xie, K. Xu, N. J. Mitra, D. Cohen-Or, W. Gong, Q. Su, and B. Chen. Sketch-to-Design: Context-Based Part Assembly. *Computer Graphics Forum*, 2013. 2

[65] B. Xu, W. Chang, A. Sheffer, A. Bousseau, J. McCrae, and K. Singh. True2form: 3d curve networks from 2d sketches via selective regularization. *ACM Trans. Graph.*, 33(4), 2014. 2

[66] K. Xu, K. Chen, H. Fu, W.-L. Sun, and S.-M. Hu. Sketch2scene: Sketch-based co-retrieval and co-placement of 3d models. *ACM Trans. Graph.*, 32(4), 2013. 2

[67] X. Yan, J. Yang, E. Yumer, Y. Guo, and H. Lee. Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision. In *Proc. NIPS*, 2016. 1, 2

[68] J. Yang, S. E. Reed, M.-H. Yang, and H. Lee. Weakly-supervised disentangling with recurrent transformations for 3d view synthesis. In *Proc. NIPS*, 2015. 1, 2

[69] R. C. Zeleznik, K. P. Herndon, and J. F. Hughes. Sketch: An interface for sketching 3d scenes. In *Proc. SIGGRAPH*, 1996. 1

[70] R. Zhang, P. Isola, and A. A. Efros. Colorful image colorization. In *Proc. ECCV*, 2016. 1, 2

[71] T. Zhou, S. Tulsiani, W. Sun, J. Malik, and A. A. Efros. View synthesis by appearance flow. In *Proc. ECCV*, 2016. 1, 2

*Additional supplementary material / results are provided on our project page: http://people.cs.umass.edu/~zlun/papers/SketchModeling/*

# 3D Shape Reconstruction from Sketches via Multi-view Convolutional Networks Supplementary Material

Zhaoliang Lun  Matheus Gadelha  Evangelos Kalogerakis  Subhransu Maji  Rui Wang
University of Massachusetts Amherst
{ zlun, mgadelha, kalo, smaji, ruiwang } @cs.umass.edu

## 1. Workflow for users

Our method can take a single or multiple sketches as input. In the case of a single sketch, our method generates the 3D shape based on the input sketch alone. Alternatively, users can provide multiple sketches as input at once, or provide them progressively while being guided by the intermediate shape reconstructions. In the latter case, illustrated in Figure 1, the workflow for users is the following: they draw from one view, then our network, which is trained to reconstruct from that view, yields a 3D shape. Users can then draw a second sketch from another view, on top of the generated shape rendered semi-transparently from that view, similar to ShadowDraw [2]. Given the previous and new line drawing as input, our network, trained to reconstruct from both views, yields an updated 3D shape. The process can continue until users are satisfied with the result, at which point they may edit the mesh directly.

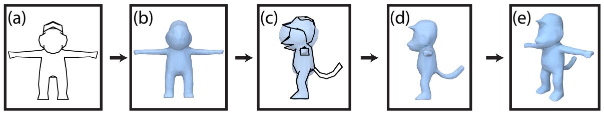

Figure 1. (a) The user can provide a front view sketch as input; (b) our network trained on a single input sketch generates an intermediate shape; (c) the user can further draw a sketch from the side view using the rendered shape as a guide; (d) & (e) our network trained on inputs from both views yields an updated 3D shape.

## 2. User study

In addition to the numerical evaluation measures in Section 4, we also performed a perceptual user study to compare our method with the volumetric decoder, view-based decoder based on Tatarchenko *et al*. [4] and the nearest neighbor sketch-based retrieval. The user study was executed through the Amazon Mechanical Turk (MTurk) service. Each questionnaire included 30 queries. Each query showed: (a) a pair of synthetic or human line drawings depicting a test shape from two different views, (b) a rendered image of the 3D surface mesh reconstructed using our method given these two input line drawings, (c) another rendered image of the 3D surface mesh reconstructed using one of the alternative methods. The images were laid out as shown in Figure 2. Queries were shown at a random order, while each page was repeated twice (i.e., 15 unique queries), with the two rendered mesh images randomly flipped, to detect unreliable users giving inconsistent answers. Each query included the following question: “Which of the two 3D models on the bottom (A or B) is MORE similar to the object depicted by the line drawings on the top? ”. Participants were asked to pick one of the following answers: “(i) A, (ii) B, (iii) can’t tell - Both A and B look equally similar to the line drawings, (iv) can’t tell - Neither A nor B looks similar to the line drawings”. To avoid any individual bias, we allowed each participant to complete only one questionnaire per category. Participants were rewarded $1 for each questionnaire completion. Each query was answered by 5 different, reliable MTurk participants. We filtered out unrealiable MTurk participants who gave two inconsistent answers to more than 7 out of the 15 unique queries in the questionnaire, or took less than 2 minutes to complete it. Participants agreed with each other 92.0% of the times, indicating a high degree of consistency across participants.

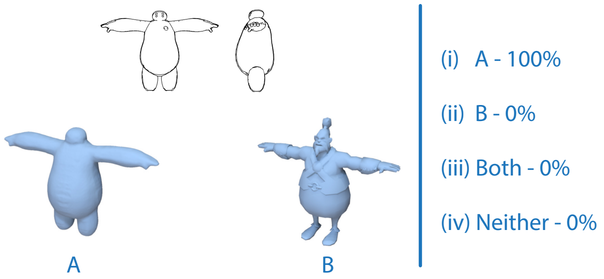

Figure 2. Query layout shown to participants of our user study.

In total, we gathered 1800 consistent responses from reliable users: 600 responses comparing the reconstructions of our method with the ones from the volumetric decoder, 600 responses comparing our method with sketch-based retrieval, and 600 responses comparing our method with the alternative view-based decoder based on [4]. The 600 query responses were gathered for all 120 human and synthetic test sketches in all our 3 categories (as explained above, each test sketch pair and resulting reconstructions was examined by 5 different, reliable MTurk participants).

Table 1 shows demographic statistics about the participants. Table 2 reports the results of the user study. We report the percentage of plurality responses per-query (plurality is formed by the 5 reliable users per query). We also re-

| | | |
|---|---|---|
| # total users | 167 | |
| # reliable users | 157 | 94.0% |
| # rejected users | 10 | 6.0% |
| # male | 104 | 62.3% |
| # female | 61 | 36.5% |
| # unknown gender | 2 | 1.2% |
| # age 18-35 | 117 | 70.1% |
| # age 36-50 | 33 | 19.8% |
| # age > 50 | 17 | 10.2% |
| # unknown age | 0 | 0.0% |
| # without post-secondary education | 26 | 15.6% |
| # with post-secondary education | 141 | 84.4% |
| # other education level | 0 | 0.0% |

Table 1. Participant statistics.

port the raw vote percentages by simply aggregating all the votes from reliable users. Table 3 shows the corresponding number of votes. Our method was found to produce shapes that look much more similar to the depicted shapes in the line drawings.

## 3. Additional evaluation

**Comparisons with variants of our method.** In addition to the evaluation described in Section 4 of our paper, we also evaluated the reconstructions produced by our method against degraded variants of it. Table 4 reports the results. Specifically, we tested the following variants: (a) we do not use the optimization procedure of Section 3.3 ('no fusion' column), (b) we set the output of our network to depth alone ('no normal' column) - since Poisson reconstruction requires both points and normals as input, we produce normals by least-squares plane fitting for each generated 3D point in this case, (c) we skip the adversarial loss term during training ('no GAN' column). For all these variants, the network uses two input sketches based on views A and B of Table 1. We also tested the reconstructions produced by our method when it uses a single sketch as input (view A, 'single input' column in Table 4) versus two sketches as input. We note that mesh fine-tuning was not used for any of these variants. Based on the resulting numbers, our full method tends to produce lower errors than its degraded variants, especially for man-made objects that often have more structural and geometric variability than character models. We also observe that using two sketches significantly improves the reconstructed shapes. This is not surprising since two input sketches contain more shape information than one.

**Evaluation with fine-tuning.** After obtaining a reconstructed mesh as described in Section 3.3, our method can further "fine-tune" the generated mesh so that it matches the input contours more precisely. We note that fine-tuning is optional, used only to add details, or "stylize", the produced meshes based on the input contours when these are precisely drawn, and if users desire so. "Fine-tuning" can be applied not only to the reconstructed meshes of our method but also to the resulting meshes of the other competing methods. Thus, we also experimented when fine-tuning is applied to the results of all methods. We found that the effect on evaluation measures tends not to be significant and our method has still much smaller errors than the others also in this case. The reason is that the mesh deformation applied during fine-tuning works well only if the produced shape matches the drawn shape in terms of structure and topology (e.g., layout and number of parts). While this is mostly true for our method, it is often not the case for shapes produced by volumetric decoders and nearest retrieval. For example, given the line drawing of a chair with a vertical middle bar on its back (Figure 4, left), the chair returned by nearest retrieval has a horizontal bar instead. Fine-tuning cannot add or remove parts, but instead deforms irrelevant surface points on the retrieved chair back towards the silhouette points of the vertical bar, yielding a largely implausible shape. Due to such mismatches, fine-tuning the retrieved shapes can slightly amplify errors with respect to ground-truth shapes. For example, for human line drawings, Hausdorff distance is further increased by 10% for nearest retrieval when fine-tuning is applied to the retrieved shapes. In contrast, for our method after fine-tuning, the error drops by a tiny amount ($< 1\%$) i.e., deformation adds small details, like the alien's eyes of Figure 1, without causing implausible deformations.

**Standard deviations.** In Table 5 and 6 we additionally provide the standard deviation of the errors for all competing methods and degraded variants of our method .

## 4. Solution to the linear system for point cloud optimization

To minimize the energy $E(\mathbf{D})$ we formulated in Section 3.3, we set its derivatives with respect to the unknown pixel depths $\mathbf{D}$ to zero, which in turn leads to a sparse linear system in the form of $\mathbf{A}\mathbf{x} = \mathbf{b}$. Here the unknown vector $\mathbf{x}$ consists of all pixel depths $d_{p,v}$ we wish to solve for. The system is solved using the conjugate gradient method in least-squares sense. Equation 1 shows the linear system along with the sparse matrix $\mathbf{A}$ and the constant vector $\mathbf{b}$. In the following paragraphs, we explain how to derive the system based on the linear constraints originating from each of the energy terms explained in Section 3.3.

$$\begin{bmatrix} w_1 I \\ \cdots \\ \left(w_2 \cdot \mathbf{n}^{(z)}_{p,v}\right) \mathbf{L}^{(x)} \\ \cdots \\ \left(w_2 \cdot \mathbf{n}^{(z)}_{p,v}\right) \mathbf{L}^{(y)} \\ \cdots \\ w_1 I \\ \cdots \\ \left(w_2 \cdot \mathbf{n}^{(z)}_{v'}(\mathbf{q}_{p,v})\right) \mathbf{L}^{(x)} \\ \cdots \\ \left(w_2 \cdot \mathbf{n}^{(z)}_{v'}(\mathbf{q}_{p,v})\right) \mathbf{L}^{(y)} \\ \cdots \end{bmatrix} [\mathbf{D}] = \begin{bmatrix} w_1 \cdot \tilde{d}_{p,v}(\mathbf{S}_t) \\ \cdots \\ -w_2 \cdot \kappa \cdot \mathbf{n}^{(x)}_{p,v} \\ \cdots \\ -w_2 \cdot \kappa \cdot \mathbf{n}^{(y)}_{p,v} \\ \cdots \\ w_1 \cdot d_{v'}(\mathbf{q}_{p,v}) \\ \cdots \\ -w_2 \cdot \kappa \cdot \mathbf{n}^{(x)}_{v'}(\mathbf{q}_{p,v}) \\ \cdots \\ -w_2 \cdot \kappa \cdot \mathbf{n}^{(y)}_{v'}(\mathbf{q}_{p,v}) \\ \cdots \end{bmatrix} \tag{1}$$

| | plurality | | | | | raw votes | | | |
|---|---|---|---|---|---|---|---|---|---|
| | A | B | both | neither | draw | A | B | both | neither |
| ours (A) vs Tatarchenko et al. (B) | 99.2% | 0.8% | 0.0% | 0.0% | 0.0% | 94.7% | 2.5% | 1.5% | 1.3% |
| ours (A) vs volumetric decoder (B) | 96.7% | 1.7% | 0.0% | 0.0% | 1.7% | 92.8% | 2.0% | 3.0% | 2.2% |
| ours (A) vs nearest retrieval (B) | 87.5% | 12.5% | 0.0% | 0.0% | 0.0% | 81.2% | 14.7% | 1.0% | 3.2% |

Table 2. Perceptual user study results comparing our method with baseline methods: per-query plurality responses (left) and raw vote percentages (right).

| | plurality | | | | | raw votes | | | |
|---|---|---|---|---|---|---|---|---|---|
| | A | B | both | neither | draw | A | B | both | neither |
| ours (A) vs Tatarchenko et al. (B) | 119 | 1 | 0 | 0 | 0 | 568 | 15 | 9 | 8 |
| ours (A) vs volumetric decoder (B) | 116 | 2 | 0 | 0 | 2 | 557 | 12 | 18 | 13 |
| ours (A) vs nearest retrieval (B) | 105 | 15 | 0 | 0 | 0 | 487 | 88 | 6 | 19 |

Table 3. Perceptual user study votes.

| | Man-made objects | | | | | Character models | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | full method | no fusion | no normal | no GAN | single input | full method | no fusion | no normal | no GAN | single input |
| Hausdorff distance | **0.092** | 0.102 | 0.108 | 0.107 | 0.134 | 0.089 | 0.090 | **0.088** | 0.098 | 0.113 |
| Chamfer distance | **0.015** | **0.015** | 0.017 | 0.016 | 0.020 | **0.015** | **0.015** | 0.016 | 0.016 | 0.021 |
| normal distance | **30.66** | 30.78 | 31.22 | 30.89 | 34.49 | **30.61** | 30.84 | 30.85 | 30.72 | 34.15 |
| depth map error | **0.026** | 0.027 | 0.029 | 0.028 | 0.035 | **0.018** | 0.019 | 0.020 | 0.019 | 0.026 |
| volumetric distance | **0.344** | 0.356 | 0.354 | 0.347 | 0.428 | **0.313** | 0.318 | 0.323 | 0.320 | 0.396 |

Table 4. Comparisons with variants of our method based on our evaluation measures (the lower the numbers, the better).

**Network prediction term.** It is easy to see that this term leads to constraints $d_{p,v} = \tilde{d}_{p,v}(\mathbf{S}_t)$ weighted by the parameter $w_1$. Therefore we can fill the matrix $\mathbf{A}$ with $w_1$'s and the vector $\mathbf{b}$ with $w_1 \cdot \tilde{d}_{p,v}(\mathbf{S}_t)$, as shown in Equation 1.

**Orthogonality term.** Considering the two orthogonality terms separately, we have two linear constraints weighted by the parameter $w_2$:

$$\mathbf{n}^{(z)}_{p,v} \cdot \frac{\partial d_{p,v}}{\partial x} = -\kappa \cdot \mathbf{n}^{(x)}_{p,v}$$

$$\mathbf{n}^{(z)}_{p,v} \cdot \frac{\partial d_{p,v}}{\partial y} = -\kappa \cdot \mathbf{n}^{(y)}_{p,v}$$

Here the superscripts $(x)$, $(y)$ and $(z)$ of the normal $\mathbf{n}_{p,v}$ indicate its $x$, $y$, or $z$ component respectively. The first-order derivatives of the depth are approximated with a gradient filter [3], which is convolved with depths in the $3 \times 3$ neighborhood per pixel:

$$\frac{\partial \mathbf{D}}{\partial x} \approx \mathbf{L}^{(x)}\mathbf{D} = \mathbf{D} * \frac{1}{12} \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -4 & 0 & 4 \\ \hline -1 & 0 & 1 \\ \hline \end{array}$$

$$\frac{\partial \mathbf{D}}{\partial y} \approx \mathbf{L}^{(y)}\mathbf{D} = \mathbf{D} * \frac{1}{12} \begin{array}{|c|c|c|} \hline 1 & 4 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -4 & -1 \\ \hline \end{array}$$

where $\mathbf{L}^{(x)}$ and $\mathbf{L}^{(y)}$ are matrices which implement the above convolution. Therefore for each pixel we can fill the corresponding columns in the sparse matrix $\mathbf{A}$ and entries in $\mathbf{b}$, as shown in the linear system of Equation 1 above.

**View consistency term.** The view consistency terms yield similar linear constraints as above. The only difference is that they use the projected depths $d_{v'}(\mathbf{q}_{p,v})$ and transformed normals $\mathbf{n}_{v'}(\mathbf{q}_{p,v})$ (instead of the depths $\tilde{d}_{p,v}(\mathbf{S}_t)$ and normals $\mathbf{n}_{p,v}$).

By combining all linear constraints, weighted by their corresponding weights, we form the overconstrained, sparse linear system of Equation 1.

## 5. More results

Our supplementary material includes rendered images of all 3D reconstructed shapes in our test data set based on our method and its degraded variants for human and synthetic line drawings in our 3 categories (‘`variant-chair.pdf`’, ‘`variant-character.pdf`’, ‘`variant-plane.pdf`’). In addition, we include rendered images of all 3D reconstructed shapes in our test data set based on our method and the alternative methods (volumetric decoder, nearest neighbor sketch-based retrieval, the view-based decoder based on Tatarchenko *et al.* and R2N2) for human and synthetic line drawings in our 3 categories (‘`baseline-chair.pdf`’, ‘`baseline-character.pdf`’,‘`baseline-plane.pdf`’). Finally, we include the MTurk participant votes for each query included in our user study (‘`user-study.pdf`’).

## References

[1] 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 *Proc. ECCV*, pages 628–644, 2016. 4

[2] Y. J. Lee, C. L. Zitnick, and M. F. Cohen. Shadowdraw: Real-time user guidance for freehand drawing. *ACM Trans. Graph.*, 30(4), 2011. 1

| | Man-made objects (synthetic) | | | | | | Character models (synthetic) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | nearest | Tatarchenko | [4]+ | volumetric | R2N2 | | nearest | Tatarchenko | [4]+ | volumetric | R2N2 |
| | ShapeMVD | retrieval | et al.[4] | U-net | decoder | [1] | ShapeMVD | retrieval | et al.[4] | U-net | decoder | [1] |
| Hausdorff distance | 0.039 | 0.064 | 0.054 | 0.060 | 0.038 | 0.073 | 0.035 | 0.091 | 0.039 | 0.033 | 0.059 | 0.049 |
| Chamfer distance | 0.006 | 0.009 | 0.007 | 0.007 | 0.006 | 0.011 | 0.005 | 0.016 | 0.008 | 0.006 | 0.007 | 0.010 |
| normal distance | 6.54 | 7.54 | 7.63 | 7.00 | 5.02 | 4.92 | 6.32 | 7.45 | 7.64 | 6.61 | 2.14 | 2.43 |
| depth map error | 0.012 | 0.018 | 0.013 | 0.014 | 0.011 | 0.014 | 0.008 | 0.015 | 0.011 | 0.008 | 0.009 | 0.010 |
| volumetric distance | 0.145 | 0.173 | 0.154 | 0.160 | 0.137 | 0.134 | 0.202 | 0.203 | 0.207 | 0.207 | 0.210 | 0.190 |
| | Man-made objects (human drawing) | | | | | | Character models (human drawing) | | | | | |
| | | nearest | Tatarchenko | [4]+ | volumetric | R2N2 | | nearest | Tatarchenko | [4]+ | volumetric | R2N2 |
| | ShapeMVD | retrieval | et al.[4] | U-net | decoder | [1] | ShapeMVD | retrieval | et al.[4] | U-net | decoder | [1] |
| Hausdorff distance | 0.081 | 0.063 | 0.054 | 0.061 | 0.039 | 0.046 | 0.061 | 0.070 | 0.062 | 0.061 | 0.054 | 0.048 |
| Chamfer distance | 0.006 | 0.013 | 0.008 | 0.007 | 0.006 | 0.007 | 0.007 | 0.014 | 0.007 | 0.007 | 0.010 | 0.012 |
| normal distance | 5.53 | 5.59 | 4.26 | 4.24 | 2.28 | 4.10 | 6.46 | 8.72 | 4.49 | 5.96 | 2.39 | 2.39 |
| depth map error | 0.009 | 0.014 | 0.011 | 0.010 | 0.009 | 0.013 | 0.010 | 0.014 | 0.011 | 0.010 | 0.012 | 0.011 |
| volumetric distance | 0.095 | 0.182 | 0.124 | 0.091 | 0.118 | 0.112 | 0.146 | 0.189 | 0.122 | 0.140 | 0.159 | 0.132 |

Table 5. Comparisons of our method with baselines based on our evaluation measures in terms of standard deviation.

| | Man-made objects | | | | | Character models | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | full method | no fusion | no normal | no GAN | single input | full method | no fusion | no normal | no GAN | single input |
| Hausdorff distance | 0.039 | 0.046 | 0.049 | 0.053 | 0.062 | 0.035 | 0.036 | 0.036 | 0.040 | 0.044 |
| Chamfer distance | 0.006 | 0.005 | 0.008 | 0.006 | 0.008 | 0.005 | 0.005 | 0.006 | 0.005 | 0.009 |
| normal distance | 6.54 | 6.43 | 6.97 | 6.68 | 6.88 | 6.32 | 6.85 | 6.45 | 6.70 | 8.04 |
| depth map error | 0.012 | 0.012 | 0.013 | 0.013 | 0.013 | 0.008 | 0.009 | 0.010 | 0.009 | 0.011 |
| volumetric distance | 0.145 | 0.166 | 0.147 | 0.149 | 0.162 | 0.202 | 0.211 | 0.204 | 0.209 | 0.213 |

Table 6. Comparisons with variants of our method based on our evaluation measures in terms of standard deviation.

[3] D. Nehab, S. Rusinkiewicz, J. Davis, and R. Ramamoorthi. Efficiently combining positions and normals for precise 3d geometry. *ACM Trans. Graph.*, 24(3), 2005. 3

[4] M. Tatarchenko, A. Dosovitskiy, and T. Brox. Multi-view 3d models from single images with a convolutional network. In *Proc. ECCV*, 2016. 1, 4