Neural Contours: Learning to Draw Lines from 3D Shapes

Difan Liu[1] Mohamed Nabail[1] Aaron Hertzmann[2] Evangelos Kalogerakis[1]
[1]University of Massachusetts Amherst [2]Adobe Research

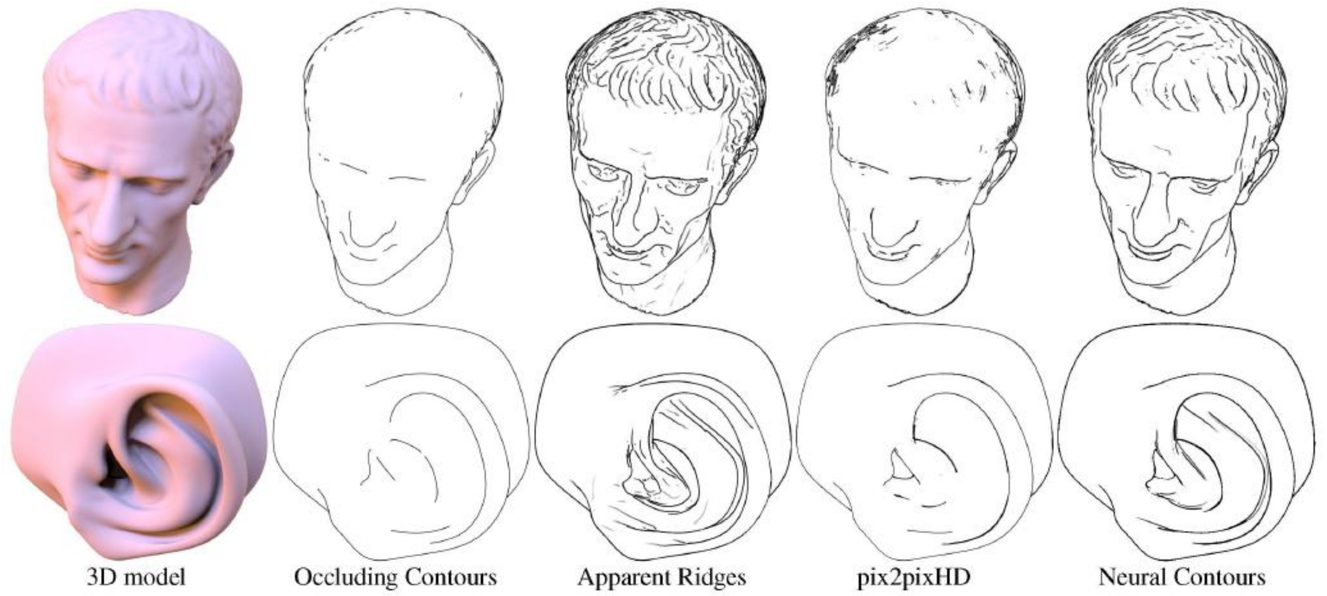


Figure 1: Given a 3D model (left), our network creates a line drawing that conveys its structure more accurately compared to using other methods individually, such as Occluding Contours [1], Apparent Ridges [22], and pix2pix variants [37]. Please see appendix for results on many more models.

Abstract

This paper introduces a method for learning to generate line drawings from 3D models. Our architecture incorporates a differentiable module operating on geometric features of the 3D model, and an image-based module operating on view-based shape representations. At test time, geometric and view-based reasoning are combined with the help of a neural module to create a line drawing. The model is trained on a large number of crowdsourced comparisons of line drawings. Experiments demonstrate that our method achieves significant improvements in line drawing over the state-of-the-art when evaluated on standard benchmarks, resulting in drawings that are comparable to those produced by experienced human artists.

1. Introduction

To draw an accurate line drawing of an object, an artist must understand the 3D shape, and select lines that will communicate that shape to a viewer. Modeling how artists perform this process remains an important open problem, intimately tied to 3D shape understanding, visual perception, and image stylization. Recent image stylization and image translation algorithms learn styles from examples, but do not take the underlying 3D geometry into account and generally do not capture outline drawing styles well. In contrast, analytic geometry-based algorithms effectively capture basic line drawing properties, and have been used in many computer animation films and games. However, these methods can still fail to capture properties of real drawings, and require parameter tuning for each individual model.

This paper presents a method for learning to generate line drawings from 3D models. Our model employs two branches. The first branch implements geometric line drawing based on suggestive contours, apparent ridges, ridges, and valleys. Existing geometric line drawing approaches employ hard-to-tune user-specified parameters that need to be determined separately for each object. In contrast, our method learns to automatically select these parameters through a differentiable module. The second branch is represented as a standard image translation network, but using view-based shape representations as input. We show that combining both branches produces the best results.

A number of challenges exist in developing such an approach. First, classic geometric lines are not readily differentiable with respect to their parameters. We combine soft thresholding functions along with image-space rendered maps of differential surface properties. Another major challenge is to combine the different geometric lines with purely learned ones. We show that a ranking module trained to assess the plausibility of the line drawing can be used to drive this combination. Finally, another challenge is to gather suitable training sets. Previous work has performed laborious in-person data collection from artists [3]. While this can provide high-quality datasets, it does not scale well, and it also produces data without much stylistic consistency. Can we train effective drawing models without employing artists to create many drawings? We describe a crowdsourcing approach to gather data using unskilled crowdworkers for ranking evaluations.

We evaluated our method based on Cole *et al.*'s [3] artist-created drawings, and find that our crowdsourced training produces state-of-the-art results. We also gathered a new extended test dataset involving more diverse 3D models following Cole *et al.*'s methodology. In all test datasets, our method generated line drawings that are significantly more accurate and perceptually more similar to artists' drawings compared to prior work, including geometric methods and image translation networks. We also conducted a MTurk evaluation study. We found that crowdworkers selected our line drawings as the best to convey a reference 3D model twice as frequently compared to the other techniques.

2. Related Work

How do artists create line drawings, and how do people perceive them? This question has been studied in art history [10], philosophy [11], neuroscience [36], and perceptual psychology [15, 24, 26]. Occluding contour algorithms are the foundation of non-photorealistic 3D computer graphics; see [1] for a survey.

Generalizations of occluding contours improved line drawing algorithms, beginning with the suggestive contours [5, 27], and continuing with Apparent Ridges [22] and several other methods; see [4] for a survey of contour generalizations. Cole *et al.* [3] performed a thorough study, enlisting human artists to create line drawings of known 3D objects. They found that existing methods could account for the majority of human-drawn lines, but many differences remained between hand-drawn and computer-generated lines. Gryaditskaya et al. [12] collect and analyze professional illustrations of objects. While these analyses yield deep insights into the nature of hand-drawn lines, the synthesis algorithms fail to match the quality of hand-drawn lines, while also requiring several parameters to be set by a user on a case-by-case basis.

Meanwhile, learned image stylization algorithms in computer vision and computer graphics, e.g., [16, 9, 40, 19], have shown the potential of learning to capture artistic styles. However, these methods do not capture dependence on an underlying 3D shape, and often neglect 3D understanding. Few style transfer methods are capable of effective stylization with line drawings. Two exceptions are Im2Pencil [28] and Inoue et al. [17], which separate object outlines from interiors, but do not explicitly consider 3D shape. StyLit stylizes 3D models [7] but does not capture line drawings.

To date, there has been almost no previous work that learns artistic style for line drawings of 3D models. Lum and Ma [29] proposed an SVM for learning lines from interactive user feedback on a single shape. Kalogerakis *et al.* [23] proposed learning hatching styles from combinations of surface orientation fields and geometric features. Cole *et al.* [3] proposed linear combinations of geometric features and decision trees to learn line drawing parameters from a small number of examples. In contrast, we learn a model that combines existing geometric and stylization algorithms, a novel large dataset, and modern neural architectures to produce a state-of-the-art line drawing algorithm.

3. Line Drawing Model

We first describe our architecture for computing a line drawing from a 3D shape. The model takes a 3D shape and camera as input, and outputs a line drawing. The 3D shape is represented as a triangle mesh that approximates a smooth surface. The output line drawing is specified as a 2D grayscale image. Our architecture has two branches (Figure 3): a "geometry branch" that performs line drawing based on geometric features of the 3D model, and an "image translation branch" that learns lines through image-to-image translation. Parameters for the geometric lines are set at run-time by a "ranking module". Training for the model is described in Section 5.

3.1. Geometry branch

The first branch of our model is based on classic geometry-based line drawing definitions, namely suggestive contours, ridges, valleys, and apparent ridges. Given a camera viewpoint and 3D shape, each of these formulations contributes to a grayscale pixel intensity map, which are combined to produce the final image $\mathbf{I}$. Their contributions depend on a set of thresholding parameters. Instead of setting these by hand, we introduce differentiable formulations to allow learning the thresholding parameters.

The first set of curves produced are **Occluding Contours** [1]. The model generates a binary mask $\mathbf{I}_C$ with "on" pixels at projections of occluding contours (Figure 2b), computed using the interpolated contour algorithm [16]. Occluding contours are parameter-free, and do not require any learning; they are used in all of our renderings. Another parameter-free set of lines are mesh boundaries, also rendered as a binary mask $\mathbf{I}_B$.

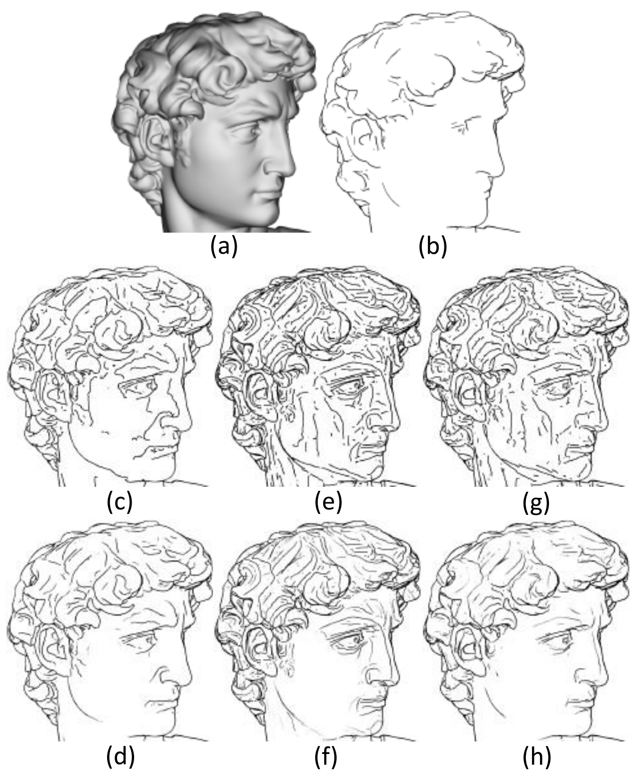


Figure 2: Given a 3D shape (a), we show (b) occluding contours, (c,d) unfiltered and filtered suggestive contours, (e,f) unfiltered and filtered ridges and valleys, (g,h) unfiltered and filtered apparent ridges.

Suggestive Contours (SCs) [5] represent surface points that are occluding contours in nearby views. See DeCarlo [5] for a detailed explanation and definitions. Let κ be the *radial curvature*, and $D\kappa$ be the directional derivative of the radial curvature at a surface point, as defined in [5]. SCs are points where $\kappa = 0$ and $D\kappa > 0$. For meshes, these curves are computed by interpolation to find the zero set of κ. As seen in Figure 2c, rendering all SCs is undesirable. Instead, "weak" SCs are filtered by only rendering SCs with $D\kappa > t_S$ for some threshold t_S, and tapered off below t_S (Figure 2d) [5]. In previous work, this t_S parameter is manually adjusted for each 3D model. In order to determine this threshold automatically, we introduce a formulation that is differentiable with respect to t_S. For a given threshold, the method outputs a per-pixel intensity map $\mathbf{I}_S$. We build two image-space maps. First, $S(\mathbf{x})$ is a binary map that is 1 at the projections of suggestive contours, and 0 otherwise, where $\mathbf{x}$ indexes pixel locations in the image. Second, $D\kappa(\mathbf{x})$ associates each pixel $\mathbf{x}$ to the directional derivative of the radial curvature at the surface point visible from that pixel. Figure 3 shows these two image-space maps for an input 3D shape. Then, the SC image is computed for each pixel $\mathbf{x}$ as:

$$\mathbf{I}_S(\mathbf{x}, t_S) = S(\mathbf{x})\ \max\left(1 - \frac{t_S}{D\kappa(\mathbf{x})},\ 0\right) \tag{1}$$

The second term filters out lines with small $D\kappa$. For $t_S = 0$, all suggestive contours are displayed, while, as t_S increases, they are eliminated. The inverse function is used rather than a linear dependence, e.g., $\max(D\kappa(\mathbf{x}) - t_S, 0)$, to produce a sharper tapering, following the implementation in `rtsc` [34]. DeCarlo *et al.* [5] also proposed filtering according to the radial direction magnitude, but we did not find that it was much different.

Ridges and Valleys (RVs) are viewpoint-independent surface extrema; see [30] for a detailed explanation. As with SCs, we introduce a formulation that is differentiable with respect to the filtering function. We introduce a threshold for ridges (t_R) and one for valleys (t_V). The per-pixel intensity maps showing locations of ridges and valleys are generated as $R(\mathbf{x})$ and $V(\mathbf{x})$, along with maps $\kappa_1(\mathbf{x}), \kappa_2(\mathbf{x})$ containing the two principal curvatures of the surface point visible from each pixel, respectively. Ridges and valleys are then filtered as:

$$\mathbf{I}_R(\mathbf{x}, t_R) = R(\mathbf{x})\ \max\left(1.0 - \frac{t_R}{\kappa_1(\mathbf{x})},\ 0.0\right) \tag{2}$$

$$\mathbf{I}_V(\mathbf{x}, t_V) = V(\mathbf{x})\ \max\left(1.0 - \frac{t_V}{\kappa_2(\mathbf{x})},\ 0.0\right) \tag{3}$$

The interpretation of the formula is similar to SCs and yields RVs consistent with `rtsc` [34]. Figures 2e and 2f show an example of RVs before and after filtering.

Apparent Ridges (ARs) [22] are object ridges from a given camera position, e.g., object points that "stick out" to the viewer; see [22] for a detailed description. We define $A(\mathbf{x})$ as the map containing ARs, and filter by the view-dependent curvature $\kappa_t(\mathbf{x})$:

$$\mathbf{I}_A(\mathbf{x}, t_A) = A(\mathbf{x})\ \max\left(1.0 - \frac{t_A}{\kappa_t(\mathbf{x})}, 0.0\right) \tag{4}$$

Figures 2g and 2h show ARs before and after filtering.

Line drawing function. Given each of these functions, we define a combined *geometric line drawing function* $\mathbf{I}_G$ conditioned on the set of parameters $\mathbf{t} = \{t_S, t_R, t_V, t_A\}$ (we drop the pixel id $\mathbf{x}$ for clarity):

$$\mathbf{I}_G(\mathbf{t}) = \max\big(\mathbf{I}_S(t_S), \mathbf{I}_R(t_R), \mathbf{I}_V(t_V), \mathbf{I}_A(t_A), \mathbf{I}_C, \mathbf{I}_B\big) \tag{5}$$

where the $\max$ function operates per pixel independently.

Preprocessing. In a pre-processing step, we compute the curvatures required for the above lines from the input mesh. using [33]. We normalize object size so that the longest dimension is equal to 1 and the curvature quantities are divided by the their $90th$ percentile value.

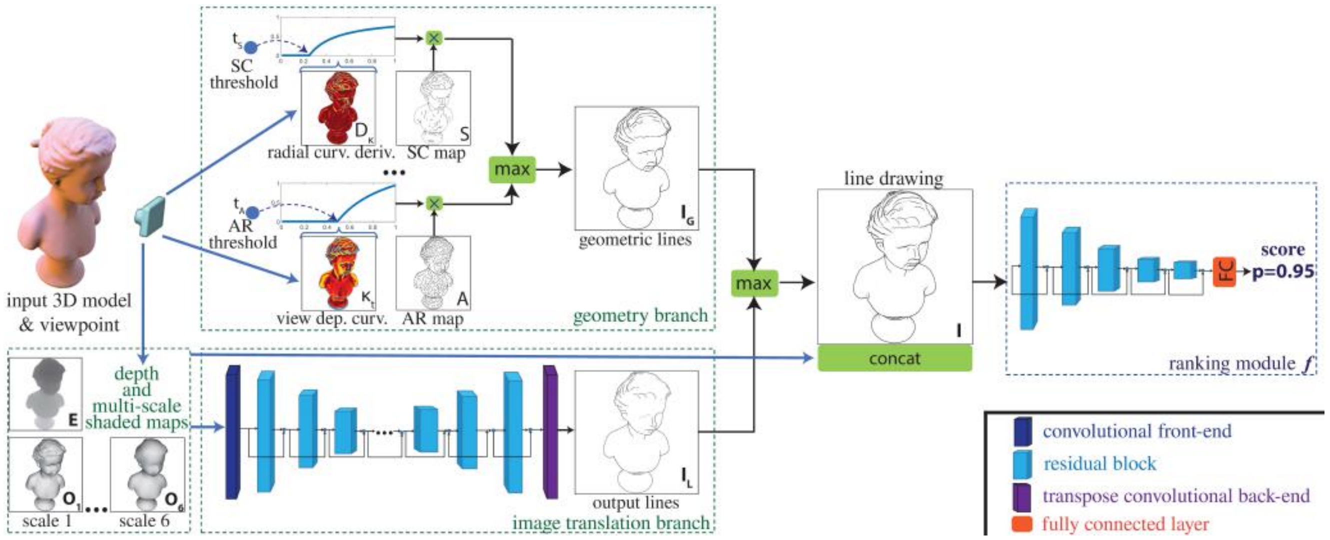


Figure 3: Our network architecture: the input 3D model is processed by a geometry branch operating on curvature features, and an image-based branch operating on view-based representations. Their outputs are combined to create a line drawing, which is in turn evaluated by a ranking module that helps determining optimal line drawing parameters.

3.2. Image translation branch

An alternative approach to create line drawings from shapes is to use a neural network that directly translates shape representations to 2D images. To simplify the mapping, one can feed view-based shape representations as input to such network (i.e., depth images, shaded renderings), which also allows us to re-purpose existing image-to-image translation networks. Our method also incorporates this generative approach. Specifically, following pix2pixHD [37], we used an image translation network module, shown in Figure 3. Details about the architecture are provided in the appendix. As input to the network, we use a combination of view-based representations. First, we use a depth image $\mathbf{E}$ of the shape from the given camera. Then we also compute shaded rendering images representing Lambertian reflectance (diffuse shading) [31] created from the dot product of surface normals with light direction (light is at the camera). We use these shaded renderings because shading features are important predictors of where people draw lines [3]. To increase robustness to rendering artifacts, we also smooth the mesh normals using diffusion [20] with different smoothing parameters (more details in the appendix). As a result, we create a stack of six smoothed versions of shaded images $\mathbf{O} = \{\mathbf{O}_1, \mathbf{O}_2, ..., \mathbf{O}_6\}$, which are concatenated channel-wise with the depth image (Figure 3, bottom left). The resolution of all images is set to 768×768. We found that using these combined multiple inputs produces better results.

We also experimented with feeding rendered curvature maps, however, as we discuss in our experiments section, the results did not improve. Finally, since the network outputs per-pixel line probabilities, we use the ridge detection procedure of Cole *et al*. [3] so that the output image $\mathbf{I}_L$ contains cleaner curves.

3.3. Neural Ranking Module

As discussed earlier, the thresholding parameters $\mathbf{t}$ play an important role in determining the existence and tapering of the geometric lines. The threshold values determine how much intensity each geometric line will contribute to the final image, if at all. Our approach determines the threshold parameters $\mathbf{t}$ at test time, since different 3D models may be best rendered by different combinations of geometric lines. We employ a Neural Ranking Module (NRM) that scores the quality of a given line drawing. Then, at test time, the thresholds $\mathbf{t}$ are set by optimization of the NRM score.

Specifically, the module is a function of the merged line drawing $\mathbf{I}(\mathbf{t}) = \max(\mathbf{I}_G(\mathbf{t}), \mathbf{I}_L)$, the depth image of the shape from the given viewpoint $\mathbf{E}$, and also the multi-scale shaded images $\mathbf{O}$ (Figure 3). The module is a neural network $f(\mathbf{I}(\mathbf{t}), \mathbf{E}, \mathbf{O}, \boldsymbol{\phi}) = p$, where p is the output score, and $\boldsymbol{\phi}$ are the learned network parameters. At test time, we aim to maximize this function (i.e., the quality of the drawing) by modifying the parameters $\mathbf{t}$:

$$\arg\max_{\mathbf{t}} f(\mathbf{I}(\mathbf{t}), \mathbf{E}, \mathbf{O}) \qquad (6)$$

The maximization is done with L-BFGS using analytic gradients $(\partial f/\partial \mathbf{I}) \cdot (\partial \mathbf{I}/\partial \mathbf{t})$, computed from backpropagation since our modules are differentiable. We also impose a non-negativity constraint on the parameters $\mathbf{t}$, as specified by the geometric definitions of the lines. To avoid local minima, we try multiple initializations of the parameter set $\mathbf{t}$ through a grid search.

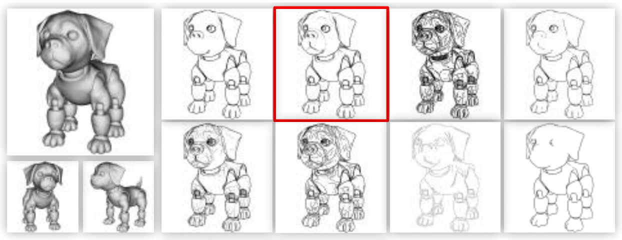
The function 6 is also used to choose whether to render mesh boundaries $\mathbf{I}_B$. This is simply a binary check that passes if the function value is higher when boundaries are included in $\mathbf{I}(t)$. Once the ranking network has determined the optimal parameters $\mathbf{t}_{opt}$, the final drawing is output as $\mathbf{I}(\mathbf{t}) = \max(\mathbf{I}_G(\mathbf{t}_{opt}), \mathbf{I}_L)$. We note that the NRM does not adjust the contribution of the image translation module at test time. We tried using a soft thresholding function on its output $\mathbf{I}_L$, but it did not help. Instead, during training, the image translation module is fine-tuned with supervisory signal from the NRM. We also experimented with directly predicting the parameters $\mathbf{t}$ with a feed-forward network, but we found that this strategy resulted in much worse performance compared to test-time optimization of $\mathbf{t}$; see the appendix for details.

NRM architecture. The neural ranking module follows the ResNet-34 architecture. The input is the line drawing $\mathbf{I}$, depth $\mathbf{E}$, and shaded maps $\mathbf{O}$ at 768×768 resolution that are concatenated channel-wise. To handle this input, we added one more residual block after the original four residual blocks of ResNet-34 to downsample the feature map by a factor of 2. The newly added residual block produces a $12 \times 12 \times 1024$ map. After mean pooling, we get a 1024-dimension feature vector. We remove the softmax layer of ResNet-34 and use a fully connected layer to output the "plausibility" value. Details about the architecture can be found in the appendix.

4. Dataset

To train the the neural ranking and image translation modules, we need a dataset of line drawings. Although there are a few large-scale human line drawing datasets available online [21, 35], the drawings are not associated to reference 3D shapes and include considerable distortions. An alternative scenario is to ask artists to provide us with line drawings depicting training 3D shapes. However, gathering a large number of human line drawings for training is labor-intensive and time-consuming. Cole *et al.*'s dataset [3] was gathered this way, and is too small on its own to train a deep model. In contrast, for each training shape, we generated multiple synthetic line drawings using `rtsc` [34] through several combinations of different lines and thresholds, and asked human subjects to select the best drawing in a relative comparison setting. Since selecting the best drawing can be subjective, we gathered votes from multiple human subjects, and used only training drawings for which there was consensus. Below we describe our dataset, then we describe the losses to train our modules.

Shape dataset. The first step to create our dataset was to select training 3D shapes from which reference line drawings will be generated. We used three collections: ShapeNet [2], Models Resource [32], and Thingi10K [39]. These


None of these line drawings are good

NEXT

Figure 4: A snapshot from our MTurk questionnaires used for gathering training line drawing comparisons. The most voted answer is highlighted as red.

shape collections contain a large variety of human-made and organic objects. In the case of ShapeNet, we sampled a random subset of up to 200 shapes from each category, to avoid category imbalance. All models have oriented ground planes specified. We removed duplicate shapes and avoided sampling low-resolution shapes with fewer than 2K faces. We also processed the meshes by correctly orienting polygons (front-facing with respect to external viewpoints), and repairing connectivity (connect geometrically adjacent but topologically disconnected polygons, weld coincident vertices). The number of shapes in all collections is 23,477.

Generating candidate line drawings. We select two random camera positions for each 3D model under the constraint that they are elevated 30 degrees from the ground plane, are aligned with the upright axis, and they point towards the centroid of the mesh (i.e., only the azimuth of each camera position is randomized). Then for each of the two camera positions, we generated 256 synthetic line drawings using all possible combinations of suggestive contours, apparent ridges, ridges with valleys under 4 thresholds (e.g. suggestive contours have 4 different thresholds [0.001, 0.01, 0.1, off]), including combinations with and without mesh creases and borders ($4 \times 4 \times 4 \times 2 \times 2 = 256$ combinations). We also generated additional synthetic line drawings using Canny edges and edge-preserving filtering [8] on rendered shaded images of shapes, each with 4 different edge detection thresholds resulting in 8 more drawings. In total, this process resulted in 264 line drawings for each shape and viewpoint. These line drawings can be similar to each other, so we selected the 8 most distinct ones by applying k-mediods ($k = 8$) and using Chamfer distance between drawn lines as metric to recover the clusters.

Questionnaires. We then created Amazon MTurk questionnaires, where, on each page, we showed 8 candidate line drawings of a 3D shape, along with rendered images of it from different viewpoints [38] (Figure 4). Each page asked human participants to select the line drawing that best conveyed the rendered shape, and was most likely to be selected by other people as well. We also provided the option not to select any drawing (i.e., if none depicted the shape reliably). We employed sentinels (drawings of irrelevant shapes) to filter unreliable participants. We had total $3,739$ reliable MTurk participants in our study (see appendix for details). For each shape and viewpoint, we gathered votes from 3 different participants. We accepted a drawing for training if it was chosen by at least two users. As a result, we gathered $21,609$ training line drawings voted as "best" per shape and viewpoint. A random subset (10% of the original dataset) was kept for hold-out validation.

5. Training

The goal of our training procedure is to learn the parameters ϕ of the neural ranking module, and the parameters $\boldsymbol{\theta}$ of the image translation branch from our training dataset, so that high-quality drawings can be generated for 3D models.

NRM training. To train the Neural Ranking Module f, we use a ranking loss based on above crowdsourced comparisons. Given a drawing $\mathbf{I}^{s,c}_{best}$ selected as "best" for shape s and viewpoint c, we generate 7 pairwise comparisons consisting of that drawing and every other drawing $\mathbf{I}^{s,c}_{other,j}$ that participated in the questionnaire. We use the hinge ranking loss to train the module [14, 6]:

$$L_R = \sum_{s,c,j} \max(m - f(\mathbf{I}^{s,c}_{best}, \mathbf{E}, \mathbf{O}, \phi) + f(\mathbf{I}^{s,c}_{other,j}, \mathbf{E}, \mathbf{O}, \phi), 0) \qquad (7)$$

where m is the margin set to 1.0.

Image translation module training. Based on the line drawings selected as "best" per reference shape and viewpoint, we use cross-entropy to train the image translation module. Specifically, we treat the intensity values $\mathbf{I}_{best}$ as target probabilities for drawing, and measure the cross-entropy of the predicted output $\mathbf{I}_L$ and $\mathbf{I}_{best}$:

$$L_{ce} = -\sum_x (\mathbf{I}_{best}(x) \log \mathbf{I}_L(x) + (1 - \mathbf{I}_{best}(x)) \log(1 - \mathbf{I}_L(x)))$$

We then further end-to-end fine-tune the image translation module along with the NRM module based on the ranking loss. We also experimented with adding the GAN-based losses of pix2pix [18] and pix2pixHD [37], yet, their main effect was only a slight sharpening of existing lines, without adding or removing any new ones.

Implementation. For the ranking module, we used the Adam optimizer [25] with learning rate $2 \cdot 10^{-5}$ and batch size 32. For the image translation module, we used Adam with learning rate set to $2 \cdot 10^{-4}$ and batch size 2. The PyTorch implementation and our dataset are available at: http://github.com/DifanLiu/NeuralContours

6. Results

We evaluate our method and alternatives quantitatively and qualitatively. To perform our evaluation, we compare synthesized line drawings with ones drawn by humans for reference shapes. Below, we describe our test datasets, evaluation measures, and comparisons with alternatives.

Test Datasets. Cole *et al*. [3] conducted a study in which artists made line drawings intended to convey given 3D shapes. The dataset contains 170 precise human line drawings of 12 3D models under different viewpoints and lighting conditions for each model. Their resolution is 1024×768 pixels. Since the number of 3D test models is small, we followed the same setup as Cole *et al*. to gather 88 more human line drawings from 3 artists for 44 3D models under two viewpoints (same resolution), including printing renderings, scanning their line drawings, and aligning them. Our new test dataset includes 13 3D animal models, 10 human body parts, 11 furniture, 10 vehicles and mechanical parts. All 3D models (including the ones from Cole *et al*.'s dataset) are disjoint from the training and validation sets.

Evaluation measures. We use precision and recall measures for comparing synthetic drawings to human-made drawings, computed in the manner proposed by Cole *et al*. [3]. Each drawing is first binarized through thinning and thresholding. Precision is defined as the fraction of drawn pixels in a synthetic drawing that are near any drawn pixel of the human drawing of the same shape under the same viewpoint. Recall is defined as the fraction of pixels in the human drawing that are near any line of the synthetic drawing. Two pixels are "near" if they are within 1mm in the coordinates of the physical page the drawing was made on; this distance was based on measurements of agreement between human drawings in Cole *et al*.'s dataset. We aggregrate precision and recall into F1-score.

We also report Intersection over Union (IoU) to measure overlap between synthetic and human drawings, based on the same definition of nearness. Lastly, we report the symmetric Chamfer distance, which measures the average distance between lines of synthetic and human drawings. Since both human and synthetic drawings include aligned silhouettes, all measures will appear artificially improved because of them. To avoid this biasing, we remove silhouettes from all human and synthetic drawings and measure performance based on the rest of the lines only.

Comparisons. We compare our method, Neural Contours (NCs), with several alternatives. (1) *Occluding Contours*. (2-4) *SC-rtsc, RV-rtsc, AR-rtsc* using `rtsc` [34] with the default thresholding parameters; occluding contours are also included in all renderings. (5) *all-rtsc* renders SCs, RVs, and ARs all together with `rtsc` [34], using the default parameters. We note that we also tried to tune these parameters using an exhaustive grid search to minimize average Chamfer distance in the training set, but this resulted in worse performance (see appendix). (6) *decision tree:* The method of Cole *et al*. [3], namely, a decision tree (M5P from Weka [13]) operating on rendered curvature and gradient maps, trained on our dataset. (7) *Canny* edges extracted from the shaded shape rendering, as suggested by [3]. The edge detection parameters are selected using grid search to minimize average Chamfer distance in our training dataset. (8) *pix2pixHD* image translation

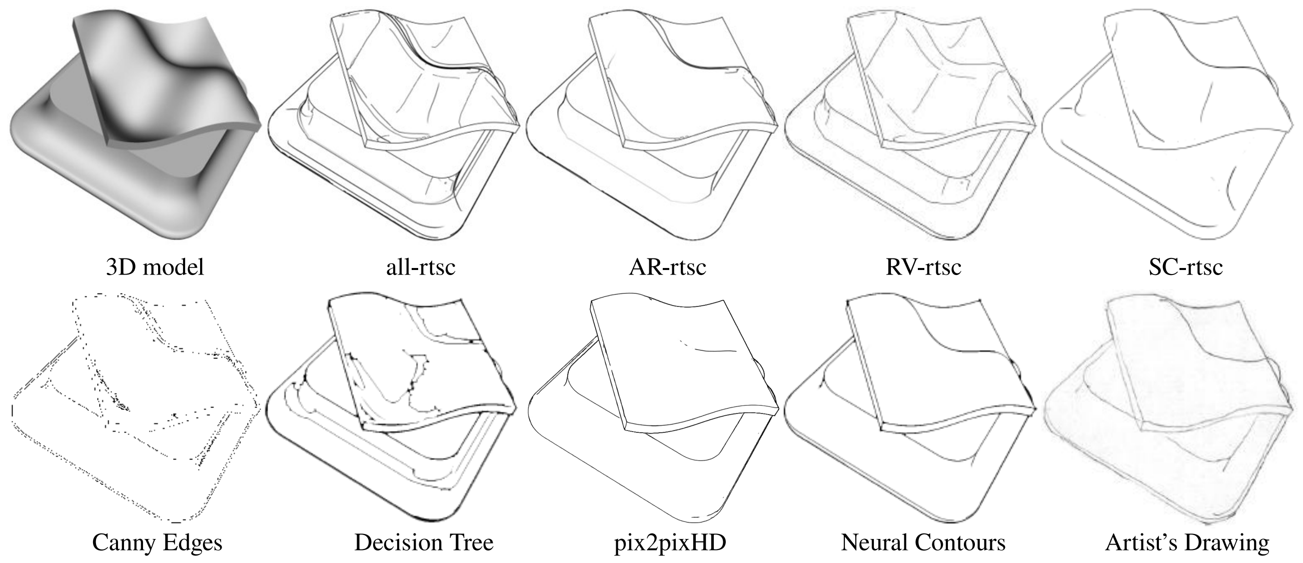


Figure 5: Comparisons with other methods. Neural Contours are more consistent with underlying shape features.

Method	IoU	CD	F1	P	R
contours	31.5	31.70	34.8	83.0	22.0
AR-rtsc	53.4	12.56	54.1	52.5	55.7
RV-rtsc	49.8	12.96	52.3	44.5	63.5
SC-rtsc	40.5	13.96	44.0	43.9	44.1
all-rtsc	48.2	12.63	52.5	40.4	75.1
Decision Tree	46.9	12.17	49.9	38.6	70.4
Canny Edges	51.9	12.59	52.9	50.4	55.8
pix2pixHD	45.0	15.73	48.7	69.6	37.5
NCs	**57.9**	**10.72**	**60.6**	60.8	60.5

Table 1: Comparisons with competing methods using all drawings from Cole *et al.*'s dataset. IoU, F1, P, R are reported in percentages, CD is pixel distance.

Method	IoU	CD	F1	P	R
contours	43.5	24.63	49.6	90.2	34.3
AR-rtsc	59.9	10.64	63.3	62.6	64.0
RV-rtsc	52.6	10.76	55.3	47.0	67.3
SC-rtsc	46.6	12.27	50.6	52.1	49.1
all-rtsc	51.8	10.74	56.5	43.8	79.6
Decision Tree	49.7	11.12	53.0	41.1	74.6
Canny Edges	58.0	11.16	61.3	56.7	66.7
pix2pixHD	50.5	13.35	54.2	75.1	42.4
NCs	**65.2**	**8.71**	**67.6**	66.3	69.0

Table 2: Comparisons with other methods using the most "consistent" human drawings from Cole *et al.*'s dataset.

Method	IoU	CD	F1	P	R
contours	49.0	19.11	54.9	92.2	39.1
AR-rtsc	66.8	9.19	69.9	69.2	70.7
RV-rtsc	64.8	9.36	66.2	62.8	70.1
SC-rtsc	65.0	9.88	63.3	61.5	65.2
all-rtsc	64.4	9.70	68.6	58.6	82.7
Decision Tree	62.1	8.93	61.1	50.9	76.6
Canny Edges	65.6	8.57	64.6	59.8	70.2
pix2pixHD	66.0	9.62	68.2	76.9	61.2
NCs	**72.4**	**7.25**	**74.6**	74.5	74.8

Table 3: Comparisons in our new test dataset.

[37], trained to output line drawings from an input depth image and shaded renderings of a shape (same as **E**, **O** in our method). Training was done on the same dataset ("best" drawings) as ours using the GAN and feature matching loss [37]. The original architecture outputs a 4096×2048 image through three local enhancer networks. In our case, since the input and output have the same resolution (1024×768), we use only the global generator of pix2pixHD.

Ablation study. We also compare with training the following reduced variants of our method. *NC-geometry* uses our geometry-based branch and our neural ranker module. *NC-image* uses the image translation module alone trained with the same losses as ours, and multi-scale shaded images as input. *NC-image-noms* uses the image translation module alone trained with the same losses as ours, and using a single shaded and depth image as input (no multi-scale shaded images). *NC-curv* is an alternative image translation module that uses curvature maps rendered in image-space concatenated with the multi-scale shaded images and depth.

Results. Tables 1 reports the evaluation measures for Cole *et al.*'s dataset for competing methods. Specifically, the synthetic drawings are compared with each human line drawing per shape and viewpoint, and the measures are averaged. Since there are artists that draw more consistently than others, we also include Table 2 as an alternative comparison. This table reports the evaluation measures in Cole *et al.*'s dataset when synthetic drawings are compared only with the most "consistent" human line drawing

Method	IoU	CD	F1	P	R
NC-geometry	60.3	10.34	64.5	76.9	55.6
NC-image	60.0	9.97	62.9	65.0	61.0
NC-image-noms	58.4	10.85	60.7	59.1	62.3
NC-image-curv	56.1	10.72	60.0	61.0	59.0
NCs	**62.8**	**9.54**	**65.4**	65.5	65.4

Table 4: Ablation study.

per shape and viewpoint, defined as the drawing that has the least Chamfer distance to the rest of the human drawings for that shape and viewpoint. We believe this comparison is more reliable than using all drawings, because in this manner, we disregard any "outlier" drawings, and also get a measure of how well methods match the most consistent, or agreeable, human line drawing. Table 3 also reports the evaluation measures for our new dataset. Based on the results, Neural Contours outperforms all competing methods in terms of IoU, Chamfer Distance, and F1, especially when we compare with the most consistent human drawings.

Table 4 reports comparisons with reduced variants of our method for the purpose of our ablation study. Our two-branch architecture offers the best performance compared to using the individual branches alone.

Figure 5 shows characteristic comparisons with competing methods, and Figure 6 shows comparisons with reduced variants of our method. We also include representative human drawings. Both figures indicate that our full method produces lines that convey shape features more similarly to what an artist would do. Figure 1 also shows comparisons with other alternatives. Our method tends to produce more accurate lines that are more aligned with underlying shape features, and with less artifacts.

Are the two branches learning the same lines? To check this hypothesis, we measure the IoU between the line drawings created from the geometry branch alone and the ones created from the image translation branch. The average IoU is 69.4%. This indicates that the two branches outputs have a partial overlap, but still they have substantial differences. As shown in Figure 6, the geometry branch makes explicit use of surface information in 3D, such as surface curvature, to identify important curves, which appear subtle or vanish in 2D rendered projections. In contrast, the image branch identifies curves that depend on view-based shading information that is not readily available in the 3D geometry (see appendix for other examples).

Which geometric lines are used more in our method? The average percentage of SCs, RVs, ARs selected by our geometry-based stylization branch are 32.2%, 16.5%, 51.3% respectively. It seems that ARs are used more dominantly, while RVs are the least frequent lines.

User study. We also conducted an Amazon MTurk study as additional perceptual evaluation. Each questionnaire page showed participants shaded renderings of a shape, along with a randomly ordered pair of synthetic drawings: one synthetic drawing from our method, and another from a different one. We asked participants which drawing best conveyed the shown 3D model. Participants could pick either drawing, specify "none", or "both" drawings conveyed the shape equally well. We asked questions twice in a random order to verify participants' reliability. We had 187 reliable participants (see appendix for details). Figure 7 summarizes the number of votes for the above options. Our method received twice the number of votes compared to the best alternative (ARs) found in this study.

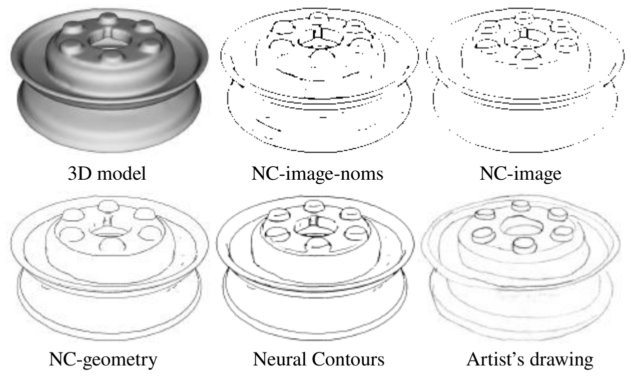


Figure 6: Comparisons with reduced NCs variants.

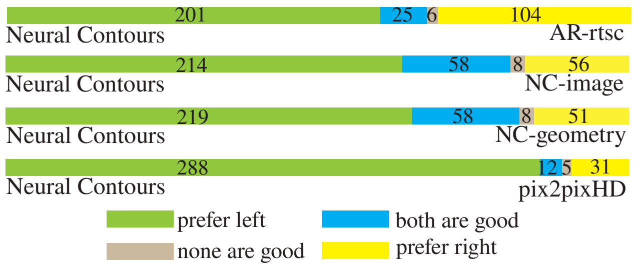


Figure 7: User study voting results.

7. Conclusion

We presented a method that learns to draw lines for 3D models based on a combination of a differentiable geometric module and an image translation network. Surprisingly, since the study by Cole *et al.* [3], there has been little progress on improving line drawings for 3D models. Our experiments demonstrate that our method significantly improves over existing geometric and neural image translation methods. There are still avenues for further improvements. Mesh artifacts (e.g., highly irregular tesselation) affect curvature estimation and shading, and in turn the outputs of both branches. Learning to repair such artifacts to increase robustness would be fruitful. Predicting drawing parameters in real-time is an open problem. Rendering the lines with learned pressure, texture, or thickness could make them match human drawings even more. Finally, our method does not handle point clouds, which would either require a mesh reconstruction step or learning to extract lines directly from unstructured point sets.

Acknowledgements. This research is partially funded by NSF (CHS-1617333) and Adobe. Our experiments were performed in the UMass GPU cluster obtained under the Collaborative Fund managed by the Massachusetts Technology Collaborative. We thank Keenan Crane for the Ogre, Snakeboard and Nefertiti 3D models.

References

[1] Pierre Bénard and Aaron Hertzmann. Line drawings from 3D models: A tutorial. *Foundations and Trends in Computer Grapics and Vision*, 11(1-2), 2019. 1, 2

[2] Angel X Chang et al. Shapenet: An information-rich 3d model repository. arXiv:1512.03012, 2015. 5

[3] Forrester Cole, Aleksey Golovinskiy, Alex Limpaecher, Heather Stoddart Barros, Adam Finkelstein, Thomas Funkhouser, and Szymon Rusinkiewicz. Where do people draw lines? *ACM Trans. Graph.*, 27(3), 2008. 2, 4, 5, 6, 8

[4] Doug DeCarlo. Depicting 3d shape using lines. In *Proc. SPIE Human Vision and Electronic Imaging XVII*, 2012. 2

[5] Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella. Suggestive contours for conveying shape. *ACM Trans. Graph.*, 22(3), 2003. 2, 3

[6] Mostafa Dehghani, Hamed Zamani, Aliaksei Severyn, Jaap Kamps, and W. Bruce Croft. Neural ranking models with weak supervision. In *Proc. SIGIR*, 2017. 6

[7] Jakub Fišer, Ondřej Jamriška, Michal Lukáč, Eli Shechtman, Paul Asente, Jingwan Lu, and Daniel Sýkora. StyLit: Illumination-guided example-based stylization of 3d renderings. *ACM Trans. Graph.*, 35(4), 2016. 2

[8] Eduardo SL Gastal and Manuel M Oliveira. Domain transform for edge-aware image and video processing. *ACM Trans. Graph.*, 30(4), 2011. 5

[9] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proc. CVPR*, 2016. 2

[10] E. H. Gombrich. *Art and Illusion: A Study in the Psychology of Pictorial Representation*. Princeton U. Press, 1961. 2

[11] Nelson Goodman. *Languages of Art: An Approach to a Theory of Symbols*. Bobbs-Merrill Company, 1968. 2

[12] Yulia Gryaditskaya, Mark Sypesteyn, Jan Willem Hoftijzer, Sylvia Pont, Frédo Durand, and Adrien Bousseau. Opensketch: A richly-annotated dataset of product design sketches. *ACM Trans. Graph.*, 38(6), 2019. 2

[13] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1), 2009. 6

[14] R. Herbrich, T. Graepel, and K. Obermayer. Support vector learning for ordinal regression. In *Proc. ICANN*, 1999. 6

[15] Aaron Hertzmann. Why do line drawings work? a realism hypothesis. *Perception*, 2020. 2

[16] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image analogies. In *Proc. SIGGRAPH*, 2001. 2

[17] Naoto Inoue, Daichi Ito, Ning Xu, Jimei Yang, Brian Price, and Toshihiko Yamasaki. Learning to trace: Expressive line drawing generation from photographs. *Computer Graphics Forum*, 38(7), 2019. 2

[18] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proc. CVPR*, 2017. 6

[19] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *Proc. ECCV*, 2016. 2

[20] Thouis R. Jones, Frédo Durand, and Mathieu Desbrun. Non-iterative, feature-preserving mesh smoothing. *ACM Trans. Graph.*, 22(3), 2003. 4

[21] J. Jongejan, H. Rowley, T. Kawashima, J. Kim, and N. Fox-Gieg. The quick, draw dataset. https://quickdraw.withgoogle.com/, 2016. 5

[22] Tilke Judd, Frédo Durand, and Edward Adelson. Apparent ridges for line drawing. *ACM Trans. Graph.*, 2007. 1, 2, 3

[23] Evangelos Kalogerakis, Derek Nowrouzezahrai, Simon Breslav, and Aaron Hertzmann. Learning hatching for pen-and-ink illustration of surfaces. *ACM Trans. Graph.*, 31(1), 2012. 2

[24] John M. Kennedy. *A Psychology of Picture Perception: Images and Information*. Jossey-Bass Publishers, 1974. 2

[25] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proc. ICLR*, 2015. 6

[26] Jan J Koenderink and Andrea J van Doorn. The shape of smooth objects and the way contours end. *Perception*, 11(2), 1982. 2

[27] Yunjin Lee, Lee Markosian, Seungyong Lee, and John F. Hughes. Line drawings via abstracted shading. *ACM Trans. Graph.*, 26(3), 2007. 2

[28] Yijun Li, Chen Fang, Aaron Hertzmann, Eli Shechtman, and Ming-Hsuan Yang. Im2pencil: Controllable pencil illustration from photographs. In *Proc. CVPR*, 2019. 2

[29] Eric B Lum and Kwan-Liu Ma. Expressive line selection by example. *The Visual Computer*, 21(8-10), 2005. 2

[30] Yutaka Ohtake, Alexander Belyaev, and Hans-Peter Seidel. Ridge-valley lines on meshes via implicit surface fitting. *ACM Trans. Graph.*, 23(3), 2004. 3

[31] Bui Tuong Phong. Illumination for computer generated pictures. *Commun. ACM*, 18(6), 1975. 4

[32] VG Resource. The models resource, https://www.models-resource.com/, 2019. 5

[33] Szymon Rusinkiewicz. Estimating curvatures and their derivatives on triangle meshes. In *Proc. 3DPVT*, 2004. 3

[34] Szymon Rusinkiewicz and Doug DeCarlo. rtsc library. http://www.cs.princeton.edu/gfx/proj/sugcon/, 2007. 3, 5, 6

[35] Patsorn Sangkloy, Nathan Burnell, Cusuh Ham, and James Hays. The sketchy database: Learning to retrieve badly drawn bunnies. *ACM Trans. Graph.*, 35(4), 2016. 5

[36] Bilge Sayim and Patrick Cavanagh. What line drawings reveal about the visual brain. *Frontiers in Human Neuroscience*, 5, 2011. 2

[37] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proc. CVPR*, 2018. 1, 4, 6, 7

[38] Michael J Wilber, Iljung S Kwak, and Serge J Belongie. Cost-effective hits for relative similarity comparisons. In *Proc. HCOMP*, 2014. 5

[39] Qingnan Zhou and Alec Jacobson. Thingi10k: A dataset of 10,000 3d-printing models. arXiv:1605.04797, 2016. 5

[40] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proc. ICCV*, 2017. 2

Appendix

1. Additional Results

Figure 8 shows a gallery of our results for various 3D models from our test set (please zoom-in to see more details). We also refer the reader to more results available on our project page: http://github.com/DifanLiu/NeuralContours

2. Image Translation Branch Implementation

We provide here more implementation details for our image translation branch (see also Section 3.2 of our main text). We also refer readers to our publicly available implementation on our project page.

Multi-scale shaded maps. We smooth mesh normals by diffusing the vertex normal field in one-ring neighborhoods of the mesh through a Gaussian distribution. Each vertex normal is expressed as a weighted average of neighboring vertex normals. The weights are set according to Gaussian functions on vertex distances. The standard deviation σ of the Gaussians control the degree of influence of neighboring vertex normals: when σ is large, the effect of smoothing is larger. The map $\mathbf{O}_1$ is generated based on the initial normal field, while $\mathbf{O}_2, \mathbf{O}_3, \mathbf{O}_4, \mathbf{O}_5, \mathbf{O}_6$ are created using smoothing based on $\sigma = \{1.0, 2.0, 3.0, 4.0, 5.0\}$ respectively.

Architecture details. Our image translation branch uses the architecture shown in Table 5. All convolutional layers are followed by batch normalization and a ReLU nonlinearity except the last convolutional layer. The last convolutional layer is followed by a sigmoid activation function. The branch contains 9 identical residual blocks, where each residual block contains two 3×3 convolutional layers with the same number of filters for both layers.

Layer	Activation size
Input	768 × 768 × 7
Conv(7x7, 7→64, stride=1)	768 × 768 × 64
Conv(3x3, 64→128, stride=2)	384 × 384 × 128
Conv(3x3, 128→256, stride=2)	192 × 192 × 256
Conv(3x3, 256→512, stride=2)	96 × 96 × 512
Conv(3x3, 512→1024, stride=2)	48 × 48 × 1024
9 Residual blocks	48 × 48 × 1024
Conv(3x3, 1024→512, stride=1/2)	96 × 96 × 512
Conv(3x3, 512→256, stride=1/2)	192 × 192 × 256
Conv(3x3, 256→128, stride=1/2)	384 × 384 × 128
Conv(3x3, 128→64, stride=1/2)	768 × 768 × 64
Conv(7x7, 64→1, stride=1)	768 × 768 × 1

Table 5: Architecture of the Image Translation Branch.

3. Neural Ranking Module Implementation

We provide here implementation details for our Neural Ranking Module (see also Section 3.3 of our main text).

Architecture details. Our Neural Ranking Module uses the architecture shown in Table 6. It follows the ResNet-34 architecture. We add one more residual block with 1024 filters after the original four residual blocks. After average pooling, we get a 1024-dimensional feature vector. We remove the softmax layer of ResNet-34 and use a fully connected layer to output the "plausibility" value.

Layer	Activation size
Input	768 × 768 × 3
Conv(7x7, 8→64, stride=2)	384 × 384 × 64
Max-pool(3x3, stride=2)	192 × 192 × 64
ResBlock(64→64, stride=1, blocks=3)	192 × 192 × 64
ResBlock(64→128, stride=2, blocks=4)	96 × 96 × 128
ResBlock(128→256, stride=2, blocks=6)	48 × 48 × 256
ResBlock(256→512, stride=2, blocks=3)	24 × 24 × 512
ResBlock(512→1024, stride=2, blocks=3)	12 × 12 × 1024
Average-pool(12x12)	1024
FC(1024→1)	1

Table 6: Architecture of the Neural Ranking Module.

4. Additional Experiments

Parameter set $\mathbf{t}$ regression. We experimented with directly predicting the parameter set $\mathbf{t}$ with a network, but this did not produce good results. The network includes a mesh encoder which is a graph neural network based on NeuroSkinning and an image encoder based on ResNet-34. The mesh encoder takes a triangular mesh as input and outputs a 1024-dimensional feature vector. The image encoder takes $(\mathbf{E}, \mathbf{O})$ as input and outputs a 1024-dimensional feature vector. These two feature vectors are concatenated and processed by a $3-$layer MLP which outputs the parameter set $\mathbf{t}$. We used cross-entropy loss between $\mathbf{I}_G(\mathbf{t})$ and $\mathbf{I}_{best}$ to train the network. We note that combining the mesh and image encoder worked the best. We name this variant *Geometry-Regressor*. Table 7 reports the resulting performance compared to our method. The results of this approach are significantly worse.

Parameter set $\mathbf{t}$ exhaustive search. We also tried to tune parameters of ARs, RVs, SCs using an exhaustive grid search to minimize average Chamfer distance in the training set. The grid was based on regular sampling 100 values of the parameters in the interval $[0, 1]$. This exhaustive search did not produce good results. Table 7 reports the performance of these variants *AR-grid*, *RV-grid*, *SC-grid*, *all-grid*.

Image translation vs geometric branch output example Figure 9 shows an additional example of comparison between the geometry branch and the image translation branch outputs; compare the areas around the antlers, and the shoulder to see the contributions of each branch.

As also shown in Figure 6 of our main paper, the geometry model makes explicit use of surface information in 3D, such as surface curvature, to identify important curves, which appear subtle or vanish in 2D rendered projections. In contrast, the image

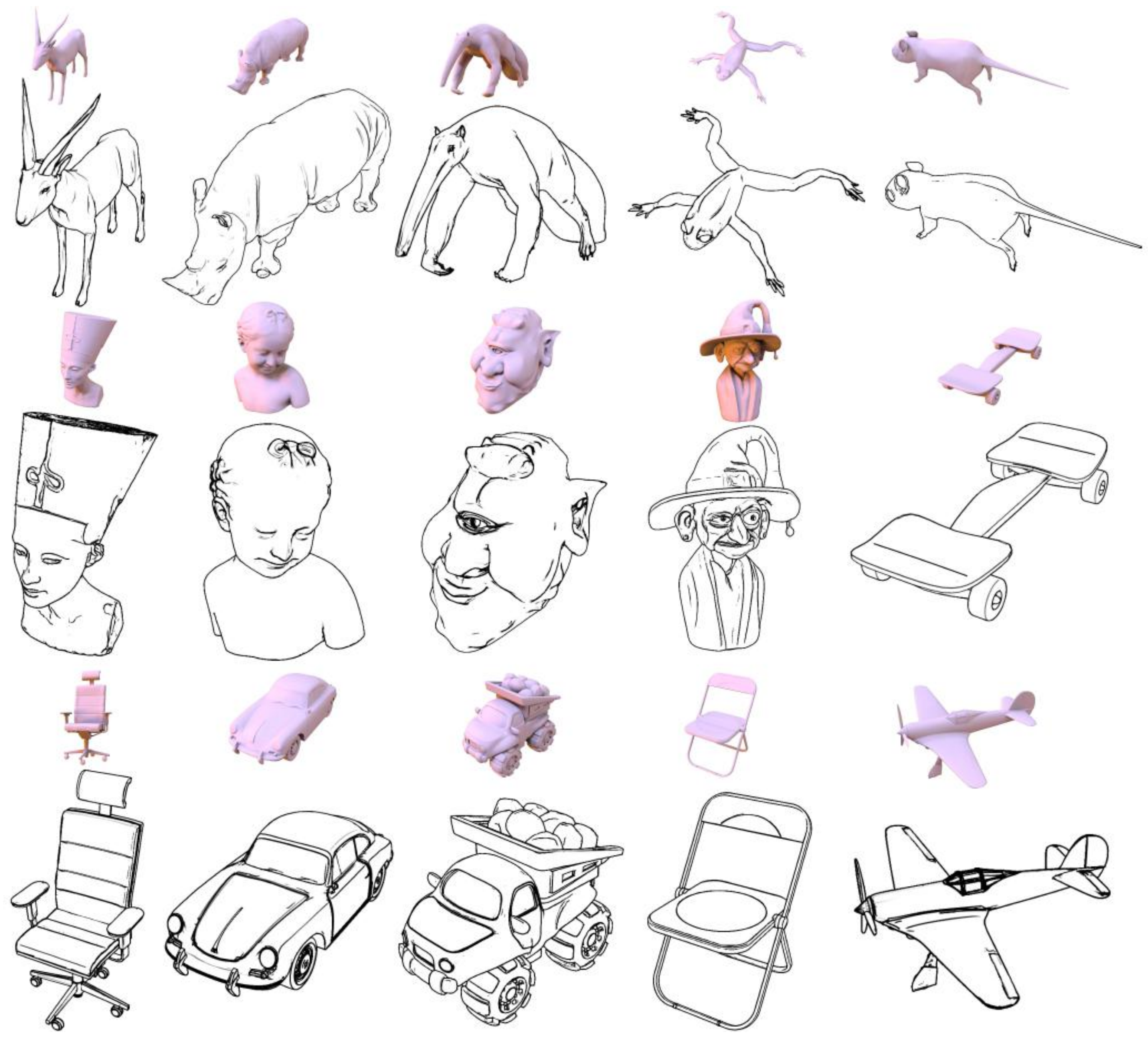


Figure 8: Results of our "Neural Contours" method on various test 3D models.

model identifies curves that depend on view-based shading information that is not readily available in the 3D geometry.

5. Training Set Collection

We created Amazon MTurk questionnaires to collect our training dataset. Each questionnaire had 35 questions. 5 of the questions were randomly chosen from a pool of 15 sentinels. Each sentinel question showed eight line drawings along with renderings from a reference 3D model. One line drawing was created by an artist for the reference shape, and seven line drawings were created for different 3D models. The line drawings were presented to the participants in a random order. Choosing one of the seven line drawings (or the option "none of these line drawings are good") resulted in failing the sentinel. If a worker failed in one of these

Method	IoU	CD	F1	P	R
AR-grid	56.6	11.21	59.1	54.2	64.9
RV-grid	56.0	11.73	58.3	53.6	63.9
SC-grid	51.0	12.57	53.2	57.5	49.5
all-grid	54.6	11.61	57.4	47.9	71.7
Geometry-Regressor	52.9	11.05	54.2	48.2	62.0
NCs	**62.8**	**9.54**	**65.4**	65.5	65.4

Table 7: Comparisons with competing methods using drawings from Cole et al.'s dataset and our newly collected dataset. IoU, F1, P, R are reported in percentages, CD is pixel distance.

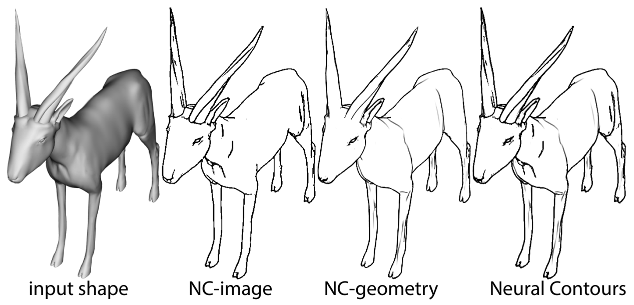


Figure 9: Additional comparison of our two branch outputs (image translation branch output "NC-Image" vs geometry branch output "NC-Geometry" vs Neural Contours).

5 sentinels, then he/she was labeled as "unreliable" and the rest of his/her responses were ignored. A total of 4396 participants took part in this user study to collect the training data. Among 4396 participants, 657 users (15%) were labeled as "unreliable". Each participant was allowed to perform the questionnaire only once.

6. Perceptual Evaluation

We conducted an Amazon Mechanical Turk perceptual evaluation where we showed participants (a) a rendered shape from a viewpoint of interest along with two more views based on shifted camera azimuth by 30 degrees, (b) a pair of line drawings placed in a randomized left/right position: one line drawing was picked from our method, while the other came from *pix2pixHD*, *NC-geometry*, *NC-image*, or *AR-rtsc*. We asked participants to select the drawing that best conveyed the shown 3D shape. Participants could pick one of four options: left drawing, right drawing, "none of the drawings conveyed the shape well", or "both" drawings conveyed the shape equally well". The study included the 12 shapes (2 viewpoints each) from both Cole *et al.*'s and our new collected test dataset (44 shapes, two viewpoints each). Thus, there were total 112 test cases, each involving the above-mentioned 4 comparisons of techniques (448 total comparisons).

Each questionnaire was released via the MTurk platform. It contained 15 unique questions, each asking for one comparison. Then these 15 questions were repeated in the questionnaire in a random order. In these repeated questions, the order of compared line drawings was flipped. If a worker gave more than 7 inconsistent answers for the repeated questions, then he/she was marked as "unreliable". Each participant was allowed to perform the questionnaire only once. A total of 225 participants took part in the study. Among 225 participants, 38 workers were marked as "unreliable". For each of the 448 comparisons, we gathered consistent answers from 3 different users. The results are shown in Figure 7 of the main text.