Neural Contours: Learning to Draw Lines from 3D Shapes
Supplementary Material

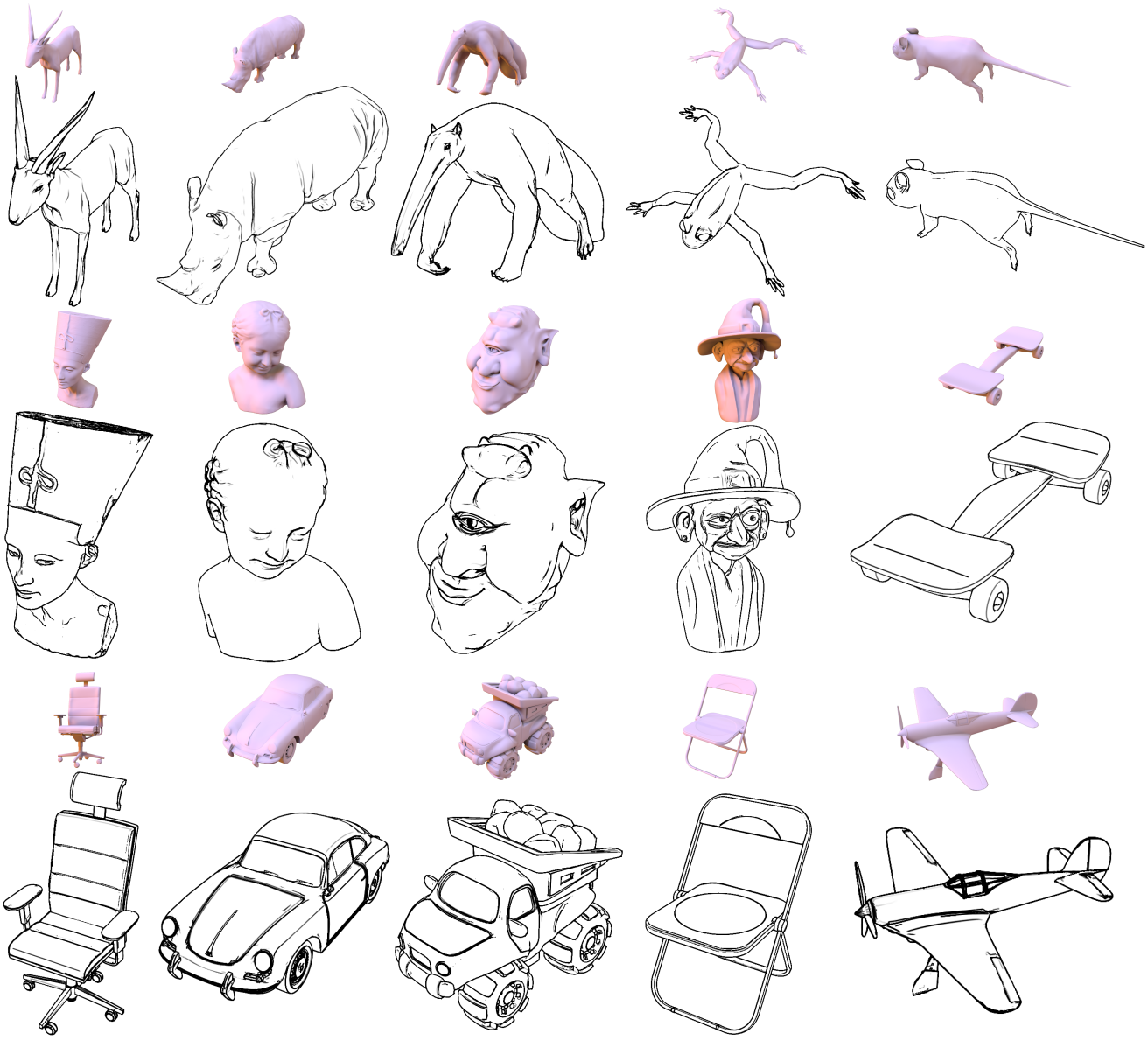

Figure 1: Results of our "Neural Contours" method on various test 3D models.

1. Additional Results

Figure 1 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,

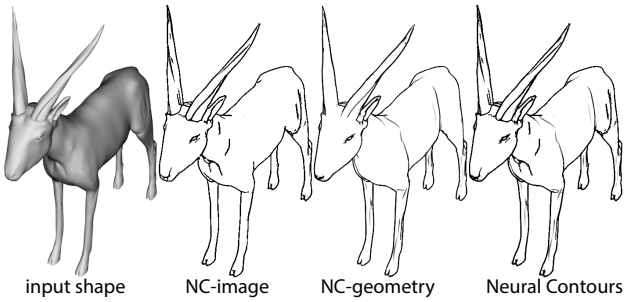

Figure 2: Additional comparison of our two branch outputs (image translation branch output "NC-Image" vs geometry branch output "NC-Geometry" vs Neural Contours).

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 1. All convolutional layers are followed by batch normalization [2] 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	$768 \times 768 \times 7$
Conv(7x7, 7→64, stride=1)	$768 \times 768 \times 64$
Conv(3x3, 64→128, stride=2)	$384 \times 384 \times 128$
Conv(3x3, 128→256, stride=2)	$192 \times 192 \times 256$
Conv(3x3, 256→512, stride=2)	$96 \times 96 \times 512$
Conv(3x3, 512→1024, stride=2)	$48 \times 48 \times 1024$
9 Residual blocks	$48 \times 48 \times 1024$
Conv(3x3, 1024→512, stride=1/2)	$96 \times 96 \times 512$
Conv(3x3, 512→256, stride=1/2)	$192 \times 192 \times 256$
Conv(3x3, 256→128, stride=1/2)	$384 \times 384 \times 128$
Conv(3x3, 128→64, stride=1/2)	$768 \times 768 \times 64$
Conv(7x7, 64→1, stride=1)	$768 \times 768 \times 1$

Table 1: 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 2. It follows the ResNet-34 architecture [1]. 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	$768 \times 768 \times 3$
Conv(7x7, 8→64, stride=2)	$384 \times 384 \times 64$
Max-pool(3x3, stride=2)	$192 \times 192 \times 64$
ResBlock(64→64, stride=1, blocks=3)	$192 \times 192 \times 64$
ResBlock(64→128, stride=2, blocks=4)	$96 \times 96 \times 128$
ResBlock(128→256, stride=2, blocks=6)	$48 \times 48 \times 256$
ResBlock(256→512, stride=2, blocks=3)	$24 \times 24 \times 512$
ResBlock(512→1024, stride=2, blocks=3)	$12 \times 12 \times 1024$
Average-pool(12x12)	1024
FC(1024→1)	1

Table 2: 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 [3] 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 3 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 3 reports the performance of these variants *AR-grid*, *RV-grid*, *SC-grid*, *all-grid*.

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 3: 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.

Image translation vs geometric branch output example Figure 2 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 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 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.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. CVPR*, 2016. 2

[2] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. ICML*, 2015. 2

[3] Lijuan Liu, Youyi Zheng, Di Tang, Yi Yuan, Changjie Fan, and Kun Zhou. Neuroskinning: Automatic skin binding for production characters with deep graph networks. *ACM Transactions on Graphics (TOG)*, 38(4), 2019. 2