Neural Strokes: Stylized Line Drawing of 3D Shapes

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

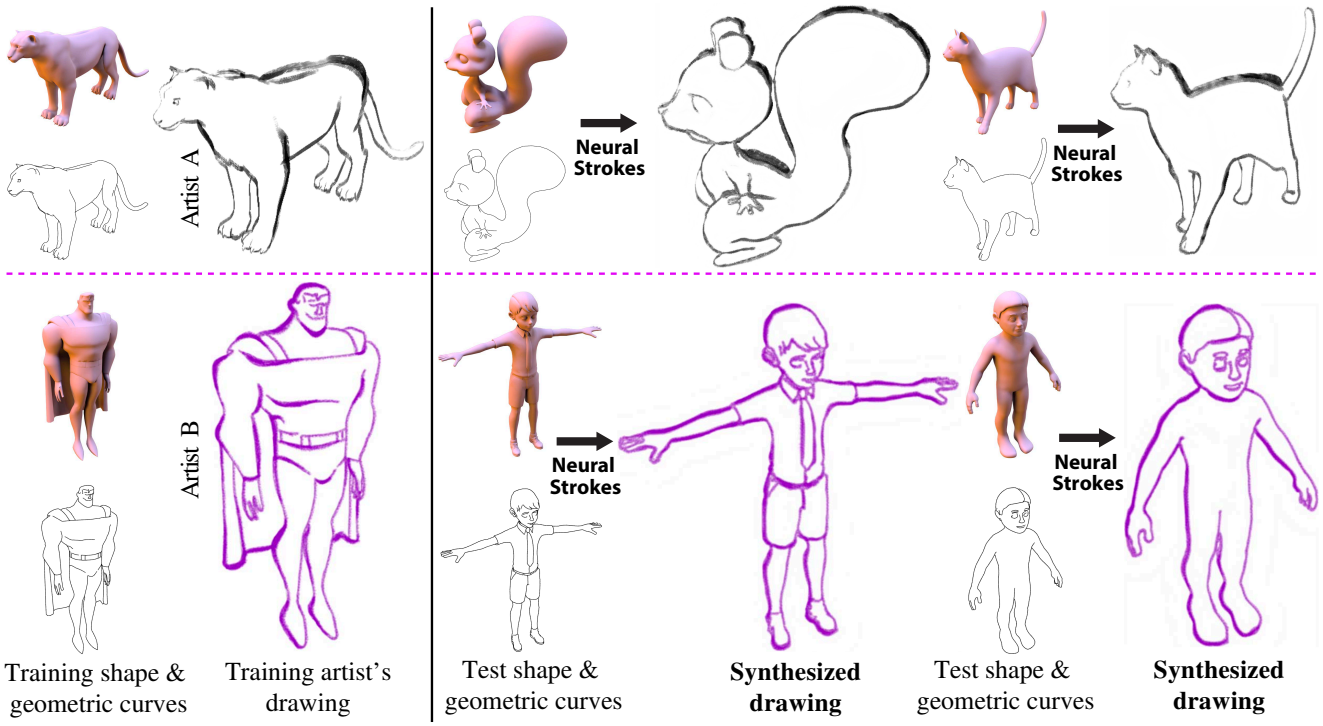

Figure 1: Our model learns to generate stylized line drawings from a single example of a training shape and corresponding drawing. Given a test 3D shape and 2D geometric curves representing the shape, our model synthesizes a line drawing in the style of the training example. Here we show synthesized drawings by transferring the artist's style A (top) or B (below).

Abstract

This paper introduces a model for producing stylized line drawings from 3D shapes. The model takes a 3D shape and a viewpoint as input, and outputs a drawing with textured strokes, with variations in stroke thickness, deformation, and color learned from an artist's style. The model is fully differentiable. We train its parameters from a single training drawing of another 3D shape. We show that, in contrast to previous image-based methods, the use of a geometric representation of 3D shape and 2D strokes allows the model to transfer important aspects of shape and texture style while preserving contours. Our method outputs the resulting drawing in a vector representation, enabling richer downstream analysis or editing in interactive applications. Our code and dataset are available at our project page: www.github.com/DifanLiu/NeuralStrokes

1. Introduction

Understanding and creating stylized outline drawings is a key task for stylization [35, 4], sketch understanding [16], and human vision [5, 19]. Artists and amateurs alike draw pictures of 3D objects in many different styles, whether for art, animation, architectural design, 3D authoring, or simply the pleasure of drawing. However, most recent research in image stylization does not take 3D geometry into account, producing drawings that frequently lose detail and do not capture image outlines. Conversely, there is a long history of 3D drawing algorithms that create precise line drawings in hand-authored procedural styles, but they cannot be learned from data, making them difficult to control and inapplicable for analysis of existing sketches. While there is a long literature on analysis and shape reconstruction from sketches, these methods typically assume that artists draw with plain line styles.

This paper introduces differentiable learning for stylization of 3D shapes, combining ideas from classic 3D line drawing algorithms with modern differentiable rendering. The algorithm produces a stylized vector rendering from a 3D shape, in a style learned from a single drawing made by an artist of a reference shape (Figure 1). There are several challenges in making such a system work. To generate stylized strokes, our method must disentangle several stroke attributes, including spatially-varying thickness, geometric deformations and smoothness, and texture. These elements may often be quite noisy, with strokes being wiggly or messy; the final pixel values are an entangled combination of these factors. Although we train from a drawing paired with a 3D shape, the individual components of thickness, deformation and texture are not labeled in the data. Training with a purely pixel-based image translation model fails to disentangle these factors, producing noisy results that lose image details. Moreover, due to the difficulties of producing skilled artistic drawings, our training set is necessarily small, with only one drawing per distinct style.

To address these challenges, we propose a differentiable rendering formulation of stroke attributes. This allows the model to learn to accurately predict stroke thickness, deformation, and texture. Because our method works with 3D geometry, it can accurately capture fine details of a shape that purely image-based methods cannot. Our model is trained on pixel inputs, yet it produces output in a vector graphics format, which provides the benefits of a compact representation with infinite resolution and is suitable for further editing and use by graphic designers.

Our technical contributions include: (a) a convolutional network operating along parameterized stroke paths, (b) the combination of 3D geometry, 2D image, and 1D curve feature maps to learn stroke properties with a differentiable vector renderer, and (c) learning from a single example based on multi-scale patches of the training drawing. We show that our method produces substantially better results than existing image-based methods, in terms of predicting artists' drawings, and in user evaluation of results.

2. Related Work

Many methods learn to stylize images from training examples. The first such methods, Image Analogies [20] and Neural Style Transfer [13], used only single-image style exemplars. Many variants of Neural Style Transfer use Gram-matrix-like losses for training or optimization from single examples, e.g., [26, 24, 34]. Other recent approaches learn stylization from larger collections of paired [25, 33] or unpaired examples [46, 40]. All of these methods take only images for input and output. However, these methods lose important geometric information, often resulting in inaccurate portrayal of shape, such as broken outlines. Moreover, these methods do not produce vector output, limiting their usefulness for certain applications.

Stylized rendering of 3D shapes has a long history in Non-Photorealistic Rendering (NPR) research [4], and these algorithms have been used in numerous applications, including movies [6], and video games [41]. Most methods entail hand-designed procedural stylization, e.g., [15, 43, 44, 29]. None of these methods can learn stylization from examples, making authoring and definition of styles challenging. Moreover, none of these methods are differentiable, making them unsuitable for integration with other vision tasks, such as sketch analysis and interpretation.

A few previous methods learn stylized 3D rendering. Bénard *et al*. [3] and StyLit [11] extend Image Analogies to stylize 3D models and animation. Neural Contours [35] learns to select which outlines to draw. Our work is complementary, since we learn to stylize outline curves. Moreover, our method produces vector rather than raster output, which is a more interpretable and useful representation.

Our work builds on ideas from learning vector strokes and stylization. Most existing methods for example-based stroke stylization [21, 37, 36, 28] are not differentiable and require vector training data. More recent methods define differentiable strokes for painting and vector graphics [12, 32], though these methods do not support texture synthesis. Our work is perhaps most similar to SketchPatch [10]. SketchPatch is an image-to-image model that translates a plain sketch to a textured sketch. However, SketchPatch does not take 3D shape and stroke geometry into consideration, and its output is a raster image, and as a result, the method often loses detail from the input geometry.

3. Model

This section describes our stylized rendering architecture. The subsequent section describes how to train the model from a single artist-drawn example. Our trained model (Figure 2) takes as input a 3D shape and a camera position and produces a stylized vector rendering $\mathbf{I}$, represented as a set of strokes, that is, curves with varying thickness and texture. This allows us to simulate the appearance of drawing media (pen, pencil, paint), and the ways artists vary pressure/thickness along strokes [17, 23].

3.1. Curve Extraction

The first stage of our model extracts geometric curves from the 3D shape, producing a set of curves $\mathbf{C}$. The goal of the rest of our model is to convert these plain curves into stylized strokes, by assigning thickness, displacement, and texture along these curves.

The curves are extracted using an existing algorithm for creating line drawings from 3D shapes. Many such methods have been developed [7], and our method can be used with any method that outputs vector curves. In this paper, we extract curves using the pretrained Geometry Branch of Neu-

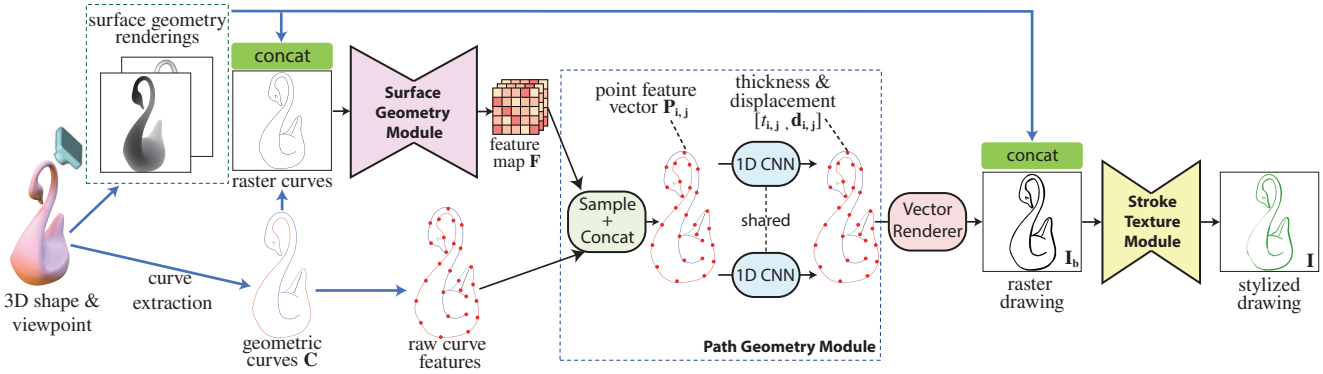

Figure 2: Our network architecture: the input 3D shape and a set of geometric curves are processed by a surface geometry module and a path geometry module to produce stroke thickness and displacement. With the predicted thickness and displacement, a stroke texture module creates a stylized line drawing with texture.

ral Contours [35], which combines curves from many prior algorithms, including Occluding Contours [4], Suggestive Contours [8], Apparent Ridges [27], Ridges & Valleys [39].

The geometric curves are represented as polylines: N vector paths $\mathbf{C} = \{\mathbf{c}_i\}_{i=1}^{N}$, where $\mathbf{c}_i$ is a sequence of densely sampled points $\mathbf{c}_i = \{\mathbf{c}_{i,j}\}_{j=1}^{M_i}$ with uniform spacing, M_i is the number of points on the path, and $\mathbf{c}_{i,j}$ represents the 2D position of point j on path i.

3.2. Stroke geometry prediction

The central portion of our model, described in this section, produces one stylized output stroke for each of the curves in $\mathbf{C}$. Texture synthesis is described in Section 3.3.

The geometric curves $\mathbf{C}$ are unstylized, and the goal of our model is to convert them to *strokes* with new shape, along with thickness and texture. The stroke control points are represented as displacements from the input curves. Displacement models the ways that artists deform curves, for example, smoothing curves, adding "wiggles" (e.g. Figure 8), and so on [9]. More precisely, for each input polyline $\mathbf{c}_i$, the module described in this section produces a 1D thickness $t_{i,j}$ for each control point $\mathbf{c}_{i,j}$, together with a displacement vector $\mathbf{d}_{i,j}$. Hence, the control points of the output stroke will be $\{\mathbf{c}_{i,j} + \mathbf{d}_{i,j}\}$ (Figure 3).

As observed in previous work, artistic stroke thickness and displacement depend on image-space geometric shape features (e.g., object depth, view-dependent curvature, and surface shading [14]). Stroke thickness and displacement also depend on the shape of the stroke itself, including phenomena like tapering, stroke smoothing, and "wiggliness," (e.g., [15]), which can be captured as deformations of the 1D curve. Hence, to predict stroke geometry, the model includes modules to incorporate information from both the shape's surface geometry and along the 1D stroke paths.

Surface geometry module. First, the surface geometry module processes surface geometry via a 2D convolutional neural network, outputting image-space feature maps $\mathbf{F}$.

Surface geometry is represented in the form of image-space renderings. Each pixel contains the geometric properties of the surface point that projects to that pixel. There are nine input channels per pixel: depth from camera, radial curvature, derivative of radial curvature [8], maximum and minimum principal surface curvatures, view-dependent surface curvature [27], dot product of surface normal with view vector, and a raster image containing the line segments of the vector paths $\mathbf{C}$. We found these geometric and shading features to be useful for predicting accurate stroke geometry. In this manner, the module jointly processes shape features and vector paths in the concatenated $768 \times 768 \times 9$ map $\mathbf{V}$. The map passes through a neural network function to output a $768 \times 768 \times 40$ deep feature map $\mathbf{F} = f(\mathbf{V}; \mathbf{w}_1)$, where f is a ResNet-based fully convolutional network [26] with four residual blocks, and $\mathbf{w}_1$ are learned during training. More details about the module architecture are given in the appendix.

Path geometry module. The path geometry module is a neural network applied separately to each input curve using 1D convolutions. Each point $\{i, j\}$ on a curve has a set of curve features and features from the shape geometry.

The curve features are 2D curve normals, 2D tangent directions, and the normalized arc length. The normalized arc length allows the model to learn to taper stroke thickness, whereas the other two features can capture image-space curve orientations. Since the orientation of the curve is ambiguous, there is a sign ambiguity in the tangent direction $\mathbf{e}_{i,j}$ and normal $\mathbf{n}_{i,j}$ per curve point. To handle the ambiguity, we extract two alternative curve features sets: one using $(\mathbf{e}_{i,j}, \mathbf{n}_{i,j})$ and another set using $(-\mathbf{e}_{i,j}, -\mathbf{n}_{i,j})$.

In addition to the curve features, the deep surface geometry features $\mathbf{F}$ generated by the surface geometry module are also included as input to the path geometry module. Specifically, for each point on a curve, we use nearest interpolation on the deep feature map $\mathbf{F}$ to produce 40-dim features. These features are concatenated with each of the

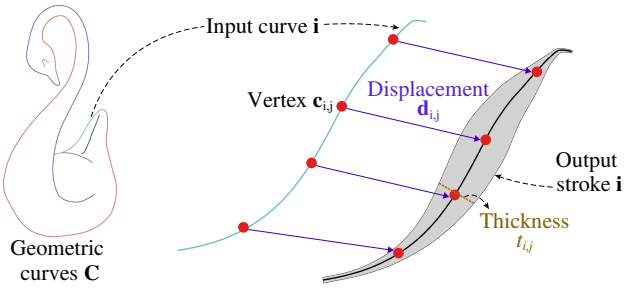

Figure 3: *Left:* an input set of geometric curves (each curve is highlighted with a different color). *Right:* For each input curve, our model outputs a stroke by predicting a thickness scalar and a 2D displacement vector for each control point.

two sets of the above 5 raw curve features of the vector path, resulting in two $M_i \times 45$ feature maps $(\mathbf{P}_i, \mathbf{P}'_i)$ for the path i, where M_i is the number of control points in the path. In this manner, the module jointly processes view-based surface features together with geometric properties specific to the path. We also experimented with processing these features independently and found the above combination yielded the best performance. The above features pass through a neural network function to predict the thickness and 2D displacement along each vector path:

$$[\mathbf{t}_i, \mathbf{d}_i] = \text{avg}\big(h(\mathbf{P}_i; \mathbf{w}_2), h(\mathbf{P}'_i; \mathbf{w}_2)\big) \qquad (1)$$

where $\mathbf{d}_i = \{\mathbf{d}_{i,j}\}_{j=1}^{M_i}$ are the predicted per-point displacements, and $\mathbf{t}_i = \{t_{i,j}\}_{j=1}^{M_i}$ are per-point thicknesses (Figure 3), and $\mathbf{w}_2$ are parameters learned from the training reference drawing. The avg function performs average pooling over predictions of the two alternative feature sets to ensure invariance to the sign of curve orientation. The function h is a 1D fully convolutional network made of 3 layers, each using filters of kernel size 3, stride 1, zero padding. The first two layers are followed by ReLU activation. The last layer has 3 output channels: two for 2D displacement, and one for thickness. For thickness, we use a ReLU activation to guarantee non-negative outputs, while for the 2D real-valued displacement output, we do not use any non-linearity.

Differentiable vector renderer. Given the predicted displacement $\mathbf{d}_i$ for each vector path $\mathbf{c}_i$, new vector paths are formed as $\mathbf{c}'_i = \mathbf{c}_i + \mathbf{d}_i$. Using the differentiable vector graphics renderer DiffVG [32], these new vector paths are rasterized into grayscale polylines based on predicted thickness $\mathbf{t}_i$. Specifically, for each pixel in the output image, its distance to the closest point on the vector paths is computed. If it is smaller than half the stroke thickness of the closest point, the pixel is inside the stroke's area and assigned black color; otherwise it is marked as white. The strokes are rendered in a 768×768 raster image $\mathbf{I}_b$ with anti-aliasing provided by the differentiable renderer. The resulting image is grayscale, lacking texture (see Figure 2).

3.3. Stroke Texture

The final module predicts texture for all strokes. Texture may vary according to depth and underlying shape features, e.g., an artist may use darker strokes for strong shape protrusions, and lighter strokes for lower-curvature regions. As a result, we condition the texture prediction not only on the raster drawing $\mathbf{I}_b$ representing the generated grayscale strokes, but also the shape representations used as input to the surface geometry module of Section 3.2. Specifically, we formulate texture prediction as a 2D image translation problem. The input to our image translation module are the first eight channels of the view-based features $\mathbf{V}$ (Section 3.2) concatenated with the raster drawing $\mathbf{I}_b$ channel-wise, resulting in a $768 \times 768 \times 9$ map $\mathbf{U}$. This map is translated into a RGB image $\mathbf{I} = g(\mathbf{U}; \mathbf{w}_3)$ where g is a ResNet-based fully convolutional network [26] with four residual blocks, and $\mathbf{w}_3$ are parameters learned during training.

As an optional post-processing step, to incorporate the predicted texture into our editable vector graphics representation, we convert the predicted RGB colors into a per stroke texture map. Specifically, each stroke is parameterized by a 2D u-v map, whose coordinates are used as a look-up table to access the texture map for each stroke. The color of each pixel in the stroke's texture map is determined by the RGB color of the corresponding pixel in the translated image $\mathbf{I}$.

4. Training

In order to train a model, we gather drawings made by artists based on rendered line drawings. Due to the difficulties in creating multiple drawings in a consistent style, our training procedure is designed to work with a single training example alone. The goal of our training procedure is to learn the parameters $\mathbf{w} = \{\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3\}$ of our surface geometry module, path geometry module, and stroke texture module described in Section 3.

Obtaining an artist's drawing. We provide an artist the feature curves $\mathbf{C}$ for shape, produced from a 3D shape using the procedure in Section 3.1. The artist is asked to produce a drawing on a digital tablet, using the provided curves as a reference. They are specifically instructed not to trace the feature curves, so that we can capture the artist's natural tendency to deform curve shape and thickness. Given the input training drawing $\hat{\mathbf{I}}$, a binary mask $\hat{\mathbf{I}}_b$ is extracted by assigning black for pixels containing the artist's strokes, and white for background. To smooth out discontinuities, anti-aliasing is applied to the mask, in the same manner as in the vector renderer, making it "soft" i.e., a grayscale image.

Note that, although we have paired drawings, i.e., input 3D geometry and a drawing, our method is not fully supervised, because the drawing is provided in raster format;

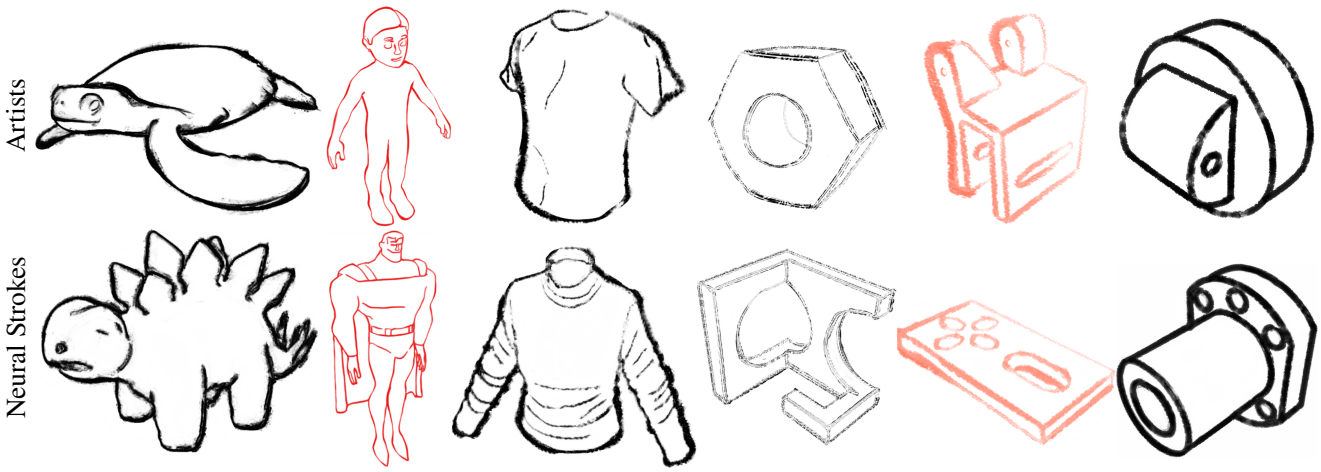

Figure 4: A gallery of our results. *Top:* artist-drawn training drawings. *Bottom:* drawings from Neural Strokes.

we do not know the stroke thickness and displacement in the drawings. This makes our data collection more flexible, allowing different data sources and allowing artists to use their favorite drawing tools.

Losses. Training a network from a single drawing is prone to overfitting. To avoid this problem, the core idea of our training procedure is to crop several random patches from the artist's drawing capturing strokes at different locations and scales. Each of the sampled patches is treated as a separate training instance. We use only patches that contain strokes. During training, we sample patches on the fly. For each patch, we randomly choose a crop size c from a set of scales $\{64 \times 64, 128 \times 128, 192 \times 192, 256 \times 256\}$, crop all images and input feature maps accordingly.

We use four terms in our loss function. First, we evaluate the cropped grayscale image $\mathbf{I}_b^c$ produced by the vector graphics renderer, as compared to the cropped reference soft mask $\hat{\mathbf{I}}_b^c$, using L_1 loss:

$$\mathcal{L}_b = ||\mathbf{I}_b^c - \hat{\mathbf{I}}_b^c||_1 \tag{2}$$

Using the above loss alone, we found that the network sometimes end up generating implausible self-intersecting and noisy strokes. To handle this problem, we add a shape regularization term on the predicted displacements:

$$\mathcal{L}_s = \frac{1}{N^c} \sum_{i=1}^{N^c} \frac{1}{(M_i^c - 1)} \sum_{j=1}^{M_i^c - 1} ||\mathbf{d}_{i,j} - \mathbf{d}_{i,j+1}||^2 \tag{3}$$

where N^c is the number of vector paths in the cropped patch and M_i^c is the number of points on the path i.

We use L_1 loss in RGB space for texture, comparing a crop $\mathbf{I}^c$ from our predicted drawing and the corresponding crop $\hat{\mathbf{I}}^c$ from the artist's drawing $\hat{\mathbf{I}}$:

$$\mathcal{L}_t = ||\mathbf{I}^c - \hat{\mathbf{I}}^c||_1 \tag{4}$$

Finally, we use an adversarial loss to encourage the output patches to be visually similar to random patches from the artist's drawing. To this end, we add a discriminator $\mathcal{D}$ during training that is trained in parallel with the stroke texture module. Architecturally, the discriminator $\mathcal{D}$ is identical to a 70×70 PatchGAN [25] with instance normalization, and it employs a standard LSGAN [38] discriminator loss. The output patches of our model are taken as *fake*, and random patches from the artist's drawing are taken as *real*. The patches are always selected to contain stroke pixels. We add the adversarial loss below to our stroke texture module by encouraging output patches to be classified as *real* by the discriminator $\mathcal{D}$:

$$\mathcal{L}_a = (\mathcal{D}(\mathbf{I}^c) - 1)^2 \tag{5}$$

Implementation details. We train the surface geometry and path geometry modules using $\lambda_b \mathcal{L}_b + \lambda_s \mathcal{L}_s$, and train the stroke texture module with $\lambda_t \mathcal{L}_t + \lambda_a \mathcal{L}_a$. The hyperparameters are set to the default values: $\lambda_b = 1, \lambda_s = 0.02, \lambda_t = 1, \lambda_a = 1$. For all three modules, we used the Adam optimizer [30] with learning rate set to 0.0002 and batch size 16.

5. Experiments

We evaluated our method both qualitatively and quantitatively. Below we discuss our dataset, evaluation metrics, comparisons with baseline methods, and our ablation study.

Dataset. To create our dataset, we collected 48 3D shapes from an online repository (TurboSquid [1]) and the ABC dataset [31], spanning several categories, including animals, humanoids, human body parts, clothes, and mechanical parts. All the 3D shapes are oriented and normalized so that the longest bounding box dimension is equal to 1. A camera position is selected for each 3D shape under the constraint that it is aligned with the upright axis and points towards the

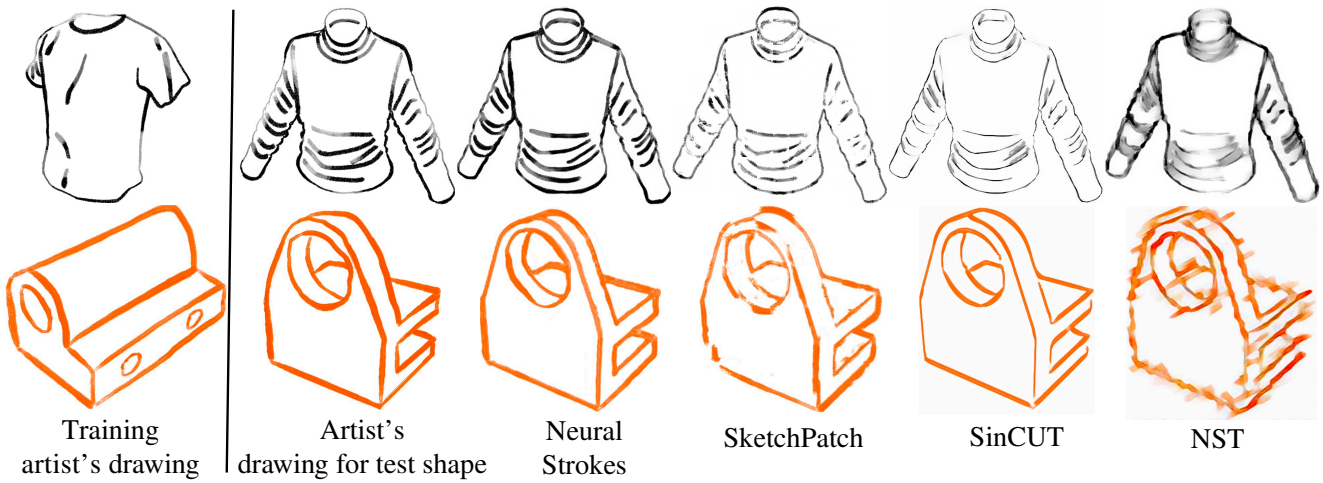

Figure 5: Comparisons with other methods. *Left to right:* training artist's drawing, artist's drawing for test shape, Neural Strokes, SketchPatch, SinCUT, NST result. Where possible, we retrained the other methods to incorporate the same geometry features present in the 3D shape as in our method. Our method produces strokes having more similar texture, intensity and thickness variation to the artist's drawing compared to other methods, which seem to miss the above style aspects.

centroid of the mesh. For each 3D shape and selected camera, a set of 2D geometric curves is extracted automatically using the geometry branch of Neural Contours [35].

We hired 12 professional artists via UpWork to stylize the 2D plain line drawings of the 3D shapes. Each artist drew with 2 to 4 different styles, resulting in 31 total styles. For each style, the artist stylized 4 plain drawings representing 4 different 3D shapes, resulting in a total of 124 drawings. Specifically, through a web questionnaire, an artist is shown each of the four 3D shapes rendered in grayscale color using a frontal view and two side views. We also provide the artist with the plain line drawing for each of the 4 shapes. The artist is explicitly instructed to use a textured brush, change the shape and vary the thickness of their strokes as they see fit to achieve their preferred style, and be stylistically consistent for all 4 drawings. Since our model is trained in a single image setting, for each style, we randomly select one drawing as training and keep the other three for testing and evaluation.

Qualitative Results. Results of our method are shown in Figure 1 and Figure 4. As shown in the leftmost image in Figure 4, our method accurately transfers variations in stroke thickness from the turtle to the dinosaur, giving thicker strokes to low-curvature regions; strokes are also thicker on right-facing parts of the surface. The method also transfers the charcoal-like stroke texture. In the second example, our method accurately transfers the thin strokes, with stroke thickness often thicker around convex bulges.

Evaluation metrics. To perform our evaluation, we compare synthesized test drawings with ones drawn by artists. We use the following metrics for evaluation: (1) **LPIPS** Learned Perceptual Image Patch Similarity [45] defined as a weighted L_2 distance between learned deep features of images. The measure has been demonstrated to correlate well with human perceptual similarity [45]. We report LPIPS averaged over all test cases in our set across all 31 styles. (2) **FID** Frechet Inception Distance [22]. FID measures the distance between two set of images in terms of statistics on deep image features. In our evaluation, the set of images includes all the synthesized line drawings of our testing set, and we compare it with the set of artists' drawings.

Comparison methods. We compare our method, Neural Strokes, with several raster image stylization approaches that attempt to transfer the style from a single example image. **(1)** ***SketchPatch*** [10] is a paired image-to-image translation model that, like ours, operates on a patch level. During training, SketchPatch takes as input the plain line drawing patches and generates stylized line drawing patches. For a fair comparison, we also condition the SketchPatch model on the input shape representations of the surface geometry module (Section 3.2) by using them as additional input channels. We also experimented with using one SketchPatch model for stroke geometry prediction and another SketchPatch model for stroke texture prediction; yet, results did not improve (see appendix). Thus, we show here the results from training a single SketchPatch model for both. **(2)** ***SinCUT*** [40] is an unpaired image-to-image translation model designed to be trained from a single image. During training of SinCUT, random crops from the training drawing are used as training instances, as in our method. We also condition the SinCUT model on the input shape representations of the surface geometry module (Section 3.2) for a fair comparison. **(3)** ***NST*** [13] performs artistic style transfer by jointly minimizing a content loss and a style loss. Given a training drawing and a testing shape, we use the test

Method	LPIPS ↓	FID ↓
SketchPatch	0.1104	83.60
SinCUT	0.1195	95.74
Bénard *et al.* [3]	0.1618	181.36
NST	0.2782	155.79
Neural Strokes	**0.0956**	**62.40**

Table 1: Numerical comparisons with other methods.

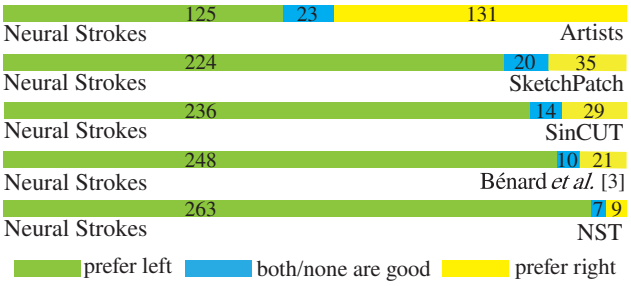

Figure 6: User study voting results.

plain line drawing as content image and the artist's training drawing as style image. **(4) Bénard *et al.*** [3] performs non-parametric line drawing stylization by copying pixel values from the artist's stylized drawing to corresponding pixels in the synthesized line drawing. The correspondence is optimized by PatchMatch [2], maximizing patch-level similarity between the reference and synthesized drawings. As pointed out by Bénard *et al.* [3], their "parameters are style specific;" i.e., one has to tune the parameters for each style. For purposes of comparison, we manually tuned their parameters to obtain the best results.

Results. Table 1 reports the evaluation measures for Neural Strokes and other competing methods. Based on the results, Neural Strokes outperforms all competing methods in terms of both LPIPS and FID. Figure 5 shows characteristic comparisons with competing methods (see also the appendix for additional comparisons). We also include the artists' drawings for the test shapes. Since other methods do not take explicit advantage of the input geometry, except in terms of the feature maps that we provided, they often introduce gaps in strokes, or blur the stroke entirely. In the top row, observe that other methods do not accurately transfer stroke thicknesses from the example. Our method produces more precise stylized strokes, with fewer artifacts, agreeing with the artists' corresponding styles in terms of stroke thickness, shape, and texture.

User study. We also conducted an Amazon MTurk study as an additional perceptual evaluation. Each questionnaire page showed participants the stylized artist's drawing for the training shape, along with a randomly ordered pair of drawings: one synthetic drawing from our method, and another from a different algorithm or from the same artist and style. We asked participants which drawing best mimicked the style of training drawing. Participants could pick either drawing, specify "none" or "both" drawings mimicked the training drawing equally well. We asked questions twice in a random order to verify participants' reliability. We had 93 reliable participants (see appendix for details and example questionnaires). Figure 6 summarizes the number of votes for the above options. The study shows that our method receives the most votes for better stylization compared to other methods, nearly seven times as many as the best alternative (SketchPatch). Moreover, our method receives similar number of votes with the artists' drawings. This indicates that our stylized drawings are comparable to artists' drawings.

Method	LPIPS ↓	FID ↓
No strokes	0.1082	75.65
No curve features	0.1006	67.74
No surface features	0.1022	72.86
No multi-scale crops	0.1056	73.70
No regularization	0.1107	71.73
Neural Strokes	**0.0956**	**62.40**

Table 2: Ablation study.

Ablation study. We also compare with the following reduced variants of our method. **(1) *No strokes:*** in this reduced variant, we remove the vector stroke representation from our method and use only the surface geometry module to predict the stroke geometry as a raster image. Specifically, the surface geometry module (a 2D fully convolutional network) takes the map $\mathbf{V}$ as input and produces the raster drawing $\mathbf{I}_b$ directly without the use of the differentiable vector graphics renderer. Since this reduced variant does not predict thickness or displacement, we remove the displacement regularization $\mathcal{L}_s$ and only use $\mathcal{L}_b$ loss for the training of surface geometry module. **(2) *No curve features:*** we remove the raw curve features from our path geometry module. Specifically, we use the surface geometry module to produce a $768 \times 768 \times 3$ map, where each pixel contains the thickness and 2D displacement prediction. Then the stroke attributes predictions are propagated to the geometric curves directly without the use of raw curve features and our 1D CNN. **(3) *No surface features:*** we exclude the 3D shape features from our path geometry module by removing the $768 \times 768 \times 8$ surface geometry renderings from the input of the surface geometry module. We also remove them from the stroke texture module. **(4) *No multi-scale crops:*** during training, instead of randomly choosing a crop size from a set of scales, we use a fixed crop size 128×128 in this variant. **(5) *No regularization:*** we remove the displacement regularization $\mathcal{L}_s$ in this variant. For all these variants, the training and architecture of stroke texture module remain the same unless specified. Table 2 reports the evaluation measures for Neural Strokes and the above-

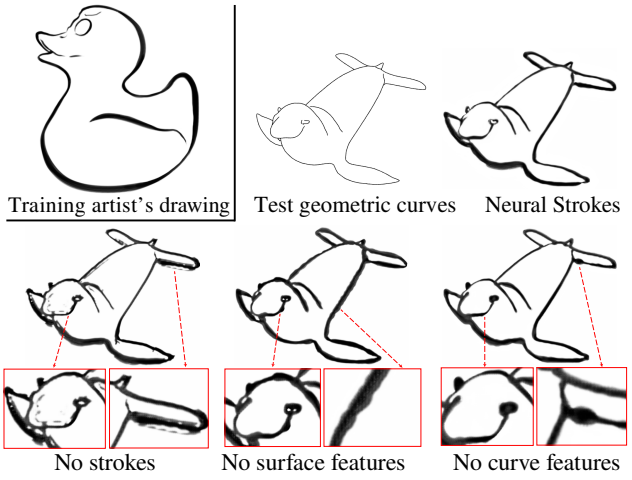

Figure 7: Comparisons with variants of our method. Removing features from our method result in noisy, incoherent strokes deviating from the training drawing style.

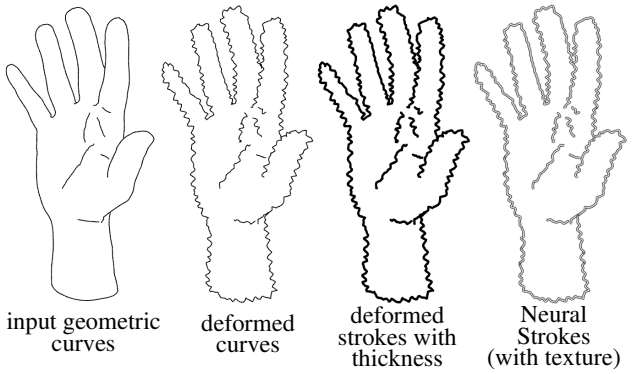

Figure 8: *Left to right:* Input geometric curves of a hand shape, deformed curves with predicted displacement from the path geometry module, strokes with predicted displacement and thickness from the path geometry module, final output textured strokes.

mentioned reduced variants. The reduced variants result in worse performance. Figure 7 shows characteristic comparisons with the reduced variants. We observe degraded results, especially in the case of "No strokes" where several broken, noisy strokes appear. In the case of "No surface features" and "No curve features", we observe incoherent strokes with unnatural thickness variation and deformation.

Intermediate results. Unlike purely raster image based methods that produce strokes as pixel values, Neural Strokes predict stroke attributes (thickness, displacement, texture) in intermediate stages that can be visualized separately. Figure 8 shows intermediate results of our method.

Vector Graphics editing. Since our method is able to output the stylized drawing in a vector representation, one can easily edit the strokes in vector graphics editing applications. In Figure 9, we show three examples of vector editing operations on our output strokes: rescaling thickness, adding wiggliness, and move control points.

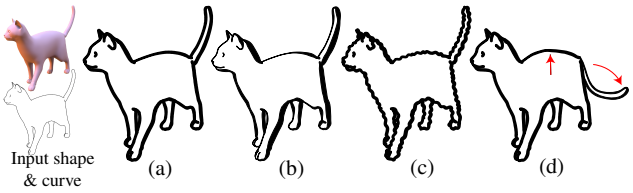

Figure 9: Given our output strokes (a) of a cat shape, we show three editing operations: (b) rescale thickness, (c) add wiggliness, (d) move control points of strokes.

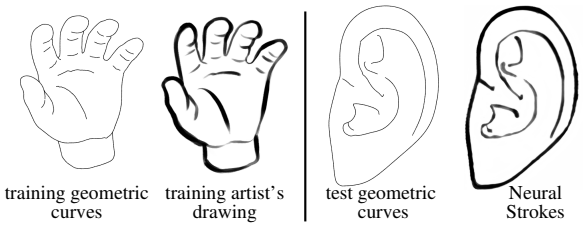

Figure 10: Example of a less successful style transfer case.

6. Conclusion

We presented a method that learns to stylize line drawings for 3D models by predicting stroke thickness, displacement and texture. The model is trained from a single raster drawing and produces output strokes in a vector graphics format. Our experiments demonstrate that our method significantly improves over existing image-based stylization methods and that our generated drawings are comparable to artists' drawing. There are still avenues for further improvements. An artist may sometimes vary the style within the same drawing, make random choices, or the style might be uncorrelated with any of the features we use. In this case, our result may not reproduce well such stylistic choices (Figure 10). In addition, when there is a large mismatch between input geometric curves and training drawing, the network may fail to reproduce correctly the stroke thickness and displacement. Learning to predict the correspondence between feature curves and the training drawing could help dealing with this issue. Learning to transfer style for other types of drawings from a single or few examples, such as hatching illustrations and cartoons, would also be another interesting research direction.

Acknowledgements. This research is partially funded by NSF (CHS-1617333) and Adobe. We thank Jonathan Eisenmann and Shayan Hoshyari for helpful discussions. We also thank the artists who contributed to our dataset.

References

[1] Turbosquid, https://www.turbosquid.com/, 2021. 5

[2] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Trans. Graph.*, 28(3), 2009. 7

[3] Pierre Bénard, Forrester Cole, Michael Kass, Igor Mordatch, James Hegarty, Martin Sebastian Senn, Kurt Fleischer, Davide Pesare, and Katherine Breeden. Stylizing animation by example. *ACM Trans. Graph.*, 32(4), 2013. 2, 7, 11, 12

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

[5] Forrester Cole, Kevin Sanik, Doug DeCarlo, Adam Finkelstein, Thomas Funkhouser, Szymon Rusinkiewicz, and Manish Singh. How well do line drawings depict shape? *ACM Trans. Graph.*, 28(3), 2009. 1

[6] Patrick Coleman, Laura Murphy, Markus Kranzler, and Max Gilbert. Making souls: Methods and a pipeline for volumetric characters. In *SIGGRAPH Talks*, 2020. 2

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

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

[9] Edgar Loy Fankbonner. *Art of Drawing the Human Body*. Inc. Sterling Publishing Co., 2004. 3

[10] Noa Fish, Lilach Perry, Amit Bermano, and Daniel Cohen-Or. Sketchpatch: sketch stylization via seamless patch-level synthesis. *ACM Trans. Graph.*, 39(6), 2020. 2, 6

[11] 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

[12] Yaroslav Ganin, Tejas Kulkarni, Igor Babuschkin, SM Ali Eslami, and Oriol Vinyals. Synthesizing programs for images using reinforced adversarial learning. In *Proc. ICML*, 2018. 2

[13] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proc. CVPR*, 2016. 2, 6

[14] Todd Goodwin, Ian Vollick, and Aaron Hertzmann. Isophote distance: A shading approach to artistic stroke thickness. In *Proc. NPAR*, 2007. 3

[15] Stéphane Grabli, Emmanuel Turquin, Frédo Durand, and François X. Sillion. Programmable rendering of line drawing from 3d scenes. *ACM Trans. Graph.*, 29(2), 2010. 2, 3

[16] Yulia Gryaditskaya, Felix Hähnlein, Chenxi Liu, Alla Sheffer, and Adrien Bousseau. Lifting freehand concept sketches into 3d. *ACM Trans. Graph.*, 39(6), 2020. 1

[17] Arthur Leighton Guptill. *Rendering in Pen and Ink*. Watson-Guptill Publications, 1997. 2

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. CVPR*, 2016. 11, 12

[19] Aaron Hertzmann. Why do line drawings work? a realism hypothesis. *Perception*, 49(4), 2020. 1

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

[21] Aaron Hertzmann, Nuria Oliver, Brian Curless, and Steven M. Seitz. Curve analogies. In *Proceedings of the 13th Eurographics Workshop on Rendering*, 2002. 2

[22] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Proc. NeurIPS*, 2017. 6

[23] Elaine Hodges. *The Guild Handbook of Scientific Illustration*. Wiley, 2003. 2

[24] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proc. ICCV*, 2017. 2

[25] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proc. CVPR*, 2017. 2, 5

[26] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *Proc. ECCV*, 2016. 2, 3, 4

[27] Tilke Judd, Frédo Durand, and Edward Adelson. Apparent ridges for line drawing. *ACM Trans. Graph.*, 26(3), 2007. 3

[28] Robert D. Kalnins, Lee Markosian, Barbara J. Meier, Michael A. Kowalski, Joseph C. Lee, Philip L. Davidson, Matthew Webb, John F. Hughes, and Adam Finkelstein. Wysiwyg npr: Drawing strokes directly on 3d models. In *Proc. SIGGRAPH*, 2002. 2

[29] Evangelos Kalogerakis, Derek Nowrouzezahrai, Patricio Simari, James McCrae, Aaron Hertzmann, and Karan Singh. Data-driven curvature for real-time line drawing of dynamic scene. *ACM Trans. Graph.*, 28(1), 2009. 2

[30] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proc. ICLR*, 2015. 5

[31] Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. Abc: A big cad model dataset for geometric deep learning. In *Proc. CVPR*, 2019. 5

[32] Tzu-Mao Li, Michal Lukáč, Michaël Gharbi, and Jonathan Ragan-Kelley. Differentiable vector graphics rasterization for editing and learning. *ACM Trans. Graph.*, 39(6), 2020. 2, 4

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

[34] Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. Universal style transfer via feature transforms. In *Proc. NeurIPS*, 2017. 2

[35] Difan Liu, Mohamed Nabail, Aaron Hertzmann, and Evangelos Kalogerakis. Neural contours: Learning to draw lines from 3d shapes. In *Proc. CVPR*, 2020. 1, 2, 3, 6

[36] Jingwan Lu, Connelly Barnes, Stephen DiVerdi, and Adam Finkelstein. Realbrush: Painting with examples of physical media. *ACM Trans. Graph.*, 32(4), 2013. 2

[37] Jingwan Lu, Fisher Yu, Adam Finkelstein, and Stephen DiVerdi. Helpinghand: Example-based stroke stylization. *ACM Trans. Graph.*, 31(4), 2012. 2

[38] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proc. ICCV*, 2017. 5

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

[40] Taesung Park, Alexei A. Efros, Richard Zhang, and Jun-Yan Zhu. Contrastive learning for unpaired image-to-image translation. In *Proc. ECCV*, 2020. 2, 6

[41] Aaron Thibault and Sean Cavanaugh. Making concept art real for borderlands. In *ACM SIGGRAPH 2010 Courses*, 2010. 2

[42] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016. 11

[43] Georges Winkenbach and David H. Salesin. Computer-generated pen-and-ink illustration. In *Proc. SIGGRAPH*, 1994. 2

[44] Georges Winkenbach and David H. Salesin. Rendering parametric surfaces in pen and ink. In *Proc. SIGGRAPH*, 1996. 2

[45] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proc. CVPR*, 2018. 6

[46] 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 comparisons and results

First, we note that our source code and dataset are available on our project website:
www.github.com/DifanLiu/NeuralStrokes

Additional comparisons with Bénard *et al.* [3]. Figure 11 shows the training artist's drawing on the top, and results from Bénard *et al.* [3] in the bottom (zoom-in for details, and compare with our results in Fig. 4, 8, 5 of our main paper). They roughly capture the overall distribution of line properties, without matching the artist's choices well. Moreover, Bénard *et al.* [3] introduces many holes and cannot handle challenging cases, such as varying stroke thickness or large deformation (the rightmost style in Figure 11).

More generalization cases. Figure 12 demonstrates challenging generalization cases: given a training drawing of a shape belonging to one category (e.g., humanoid), we synthesize a drawing for a shape from an entirely different category (e.g., mechanical object) in the same style. Our method still generalizes sufficiently in these challenging cases.

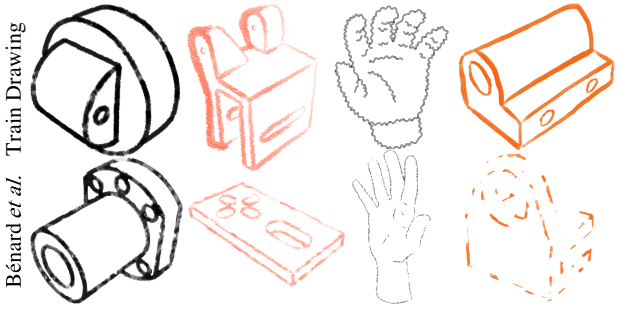

Figure 11: *Top:* artist-drawn training drawings. *Bottom:* results from Bénard *et al.* [3].

2. Network architecture

We provide here additional details of our network architecture (see also Section 3.2 and 3.3 of our main text).

Surface geometry module. Our surface geometry module uses the architecture shown in Table 3. All convolutional layers are followed by instance normalization [42] and a ReLU nonlinearity. The module contains 4 residual blocks [18], where each residual block contains two 3×3 convolutional layers with the same number of filters for both layers.

Path geometry module. Our path geometry module uses the architecture shown in Table 4. The first two convolutional layers are followed by a ReLU nonlinearity. The last layer has 3 output channels: two for 2D displacement, and one for thickness. For thickness, we use a ReLU activation

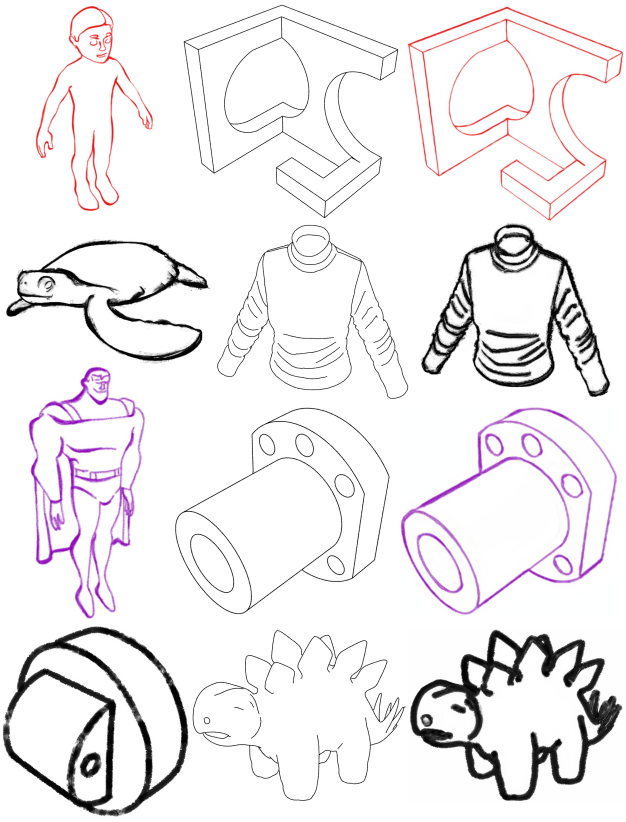

Figure 12: *Left to right:* training artist's drawing, test geometric curves, Neural Strokes.

Layer	Activation size
Input	$768 \times 768 \times 9$
Conv2D(7x7, 9→10, stride=1)	$768 \times 768 \times 10$
Conv2D(3x3, 10→20, stride=2)	$384 \times 384 \times 20$
Conv2D(3x3, 20→40, stride=2)	$192 \times 192 \times 40$
4 Residual blocks	$192 \times 192 \times 40$
Conv2D(3x3, 40→40, stride=1/2)	$384 \times 384 \times 40$
Conv2D(3x3, 40→40, stride=1/2)	$768 \times 768 \times 40$
Conv2D(1x1, 40→40, stride=1)	$768 \times 768 \times 40$

Table 3: Architecture of the surface geometry module.

to guarantee non-negative outputs, while for the 2D real-valued displacement output, we do not use any nonlinearity.

Stroke texture module. Our stroke texture module uses the architecture shown in Table 5. All convolutional layers are followed by instance normalization [42] and a ReLU nonlinearity except for the last convolutional layer. The last convolutional layer is followed by a sigmoid activa-

Layer	Activation size
Input	$M_i \times 45$
Conv1D(3x3, 45→40, stride=1)	$M_i \times 40$
Conv1D(3x3, 40→40, stride=1)	$M_i \times 40$
Conv1D(3x3, 40→3, stride=1)	$M_i \times 3$

Table 4: Architecture of the path geometry module.

Layer	Activation size
Input	$768 \times 768 \times 9$
Conv2D(7x7, 9→64, stride=1)	$768 \times 768 \times 64$
Conv2D(3x3, 64→128, stride=2)	$384 \times 384 \times 128$
Conv2D(3x3, 128→256, stride=2)	$192 \times 192 \times 256$
6 Residual blocks	$192 \times 192 \times 256$
Conv2D(3x3, 256→128, stride=1/2)	$384 \times 384 \times 128$
Conv2D(3x3, 128→64, stride=1/2)	$768 \times 768 \times 64$
Conv2D(7x7, 64→3, stride=1)	$768 \times 768 \times 3$

Table 5: Architecture of the stroke texture module.

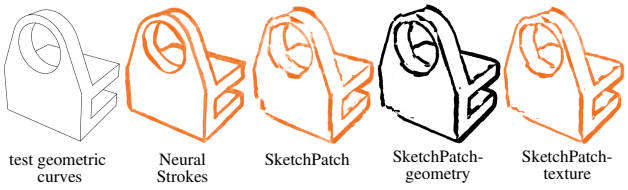

Figure 13: *Left to right:* test geometric curves, Neural Strokes, SketchPatch, SketchPatch-geometry, SketchPatch-texture result.

tion function. The module contains 6 residual blocks [18], where each residual block contains two 3×3 convolutional layers with the same number of filters for both layers.

3. Additional experiments

We experimented with using one SketchPatch model for stroke geometry prediction and another SketchPatch model for stroke texture prediction, as discussed in Section 5 of our main text ("comparison methods" paragraph). Specifically, in the first step, we train a SketchPatch model (called *SketchPatch-geometry*) on the training stroke mask $\hat{\mathbf{I}}_b$ to predict stroke geometry as a grayscale raster image. In the second step, we train another SketchPatch model (called *SketchPatch-texture*) on the training drawing $\hat{\mathbf{I}}$ to generate a stylized line drawing given the output of *SketchPatch-geometry*. The results did not improve compared to *SketchPatch* in terms of our evaluation metrics (see Table 6). Figure 13 shows example output of *SketchPatch-geometry* and *SketchPatch-texture*.

Method	LPIPS ↓	FID ↓
SketchPatch	0.1104	83.60
SketchPatch-texture	0.1142	86.96
Neural Strokes	**0.0956**	**62.40**

Table 6: Quantitative evaluation of SketchPatch variants.

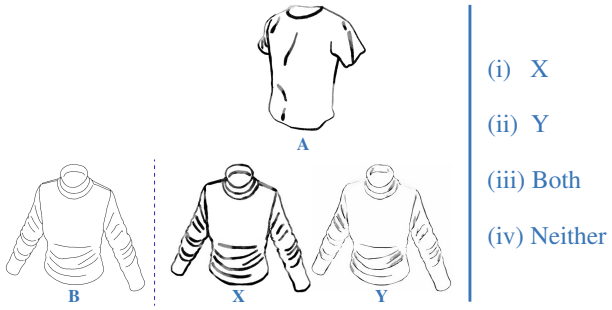

Figure 14: Layout shown to participants of our user study.

4. Perceptual evaluation

We conducted an Amazon Mechanical Turk perceptual evaluation where we showed participants (a) a stylized artist's drawing for a training shape (Figure 14, A) , (b) test geometric curves (Figure 14, B) , (c) a pair of stylized line drawings of the test shape placed in a randomized left/right position (Figure 14, X and Y): one line drawing was picked from our method, while the other came from *SketchPatch*, *SinCUT*, *NST*, Bénard *et al.* [3], or *Artists* (5 possible comparison cases). We asked participants to select the drawing that best mimicked the style of training drawing A. Participants could pick one of four options: drawing X, drawing Y, "neither of the drawings mimicked the style well", or "both drawings mimicked the style well". The study included the 31 styles from our dataset and each style consists of 3 test shapes. As a result, there were total 93 test cases, each involving the above-mentioned 5 comparisons (465 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 5 inconsistent answers for the repeated questions, then the worker was marked as "unreliable". Each participant was allowed to perform the questionnaire only once to ensure participant diversity. A total of 161 participants took part in the study. Among 161 participants, 68 workers were marked as "unreliable". For each of the 465 comparisons, we gathered votes from 3 different "reliable" users. The results are shown in Figure 6 of the main text.