Im2Pencil: Controllable Pencil Illustration from Photographs

Yijun Li[1], Chen Fang[2], Aaron Hertzmann[3], Eli Shechtman[3], Ming-Hsuan Yang[1,4]
[1]UC Merced [2]ByteDance AI [3]Adobe Research [4]Google Cloud
{yli62,mhyang}@ucmerced.edu fangchen@bytedance.com {elishe,hertzman}@adobe.com

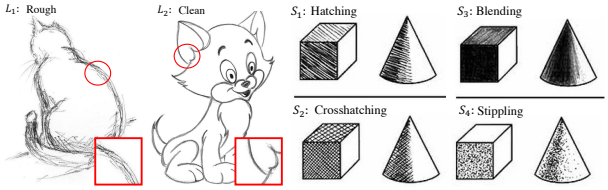
Abstract

We propose a high-quality photo-to-pencil translation method with fine-grained control over the drawing style. This is a challenging task due to multiple stroke types (e.g., outline and shading), structural complexity of pencil shading (e.g., hatching), and the lack of aligned training data pairs. To address these challenges, we develop a two-branch model that learns separate filters for generating sketchy outlines and tonal shading from a collection of pencil drawings. We create training data pairs by extracting clean outlines and tonal illustrations from original pencil drawings using image filtering techniques, and we manually label the drawing styles. In addition, our model creates different pencil styles (e.g., line sketchiness and shading style) in a user-controllable manner. Experimental results on different types of pencil drawings show that the proposed algorithm performs favorably against existing methods in terms of quality, diversity and user evaluations.

1. Introduction

Pencil is a popular drawing medium often used for quick sketching or finely-worked depiction. Notably, two main components are the *outlines* that define region boundaries, and *shading* that reflects differences in the amount of light falling on a region as well as its intensity or tone and even texture. Each of these may be applied in various different styles. For example, pencil outlines may be more or less "sketchy" (Figure 1(a)). Shading may be also more or less sketchy and use different types of hatching strategies (Figure 1(b)). Hence, we seek to accurately reproduce these drawing styles and allow users to select based on personal preferences.

We split the task into generating the outlines and shading separately, and express each as an image-to-image translation problem, learning the mapping from a collection of pencil drawings. Unfortunately, gathering paired training data for any artistic stylization task is challenging, due to the cost, as well as the spatial distortions in drawings [6]. To avoid the burden of gathering ground-truth paired data,



(a) Pencil outlines (b) Pencil shading

Figure 1. Examples of real pencil drawings in the outline ($L_1 \sim L_2$) and shading ($S_1 \sim S_4$) styles that we train on.

we instead propose to create data pairs. In particular, we filter each pencil drawing with procedures that extract outlines, tone, and edges. This produces two sets of paired data that can be used for our two subtasks, learning the outlines and shading drawing. These filters generate the same abstractions (outlines, tone, and edges) when applied to input photographs. Hence, at test-time, we filter an input photograph and then apply the trained model to produce pencil illustrations in a user-selected style.

We achieve control over different pencil styles (e.g., sketchiness and shading) by training the model with several distinct styles, where the style label is provided as an input selection unit. Moreover, the filtering modules contain additional controllable parameters to generate different abstracted inputs, which are then mapped to pencil drawings with different characteristics. On the other hand, we find that image translation architectures can often produce undesirable hatching patterns. We describe these issues and show how careful design of network architectures can address these problems.

A long-term goal of this research direction is to provide more fine-grained control to neural-network stylization algorithms. Existing learning-based methods do not provide much control over style except by changing the training input. In contrast, classical procedural stylization algorithms can provide many different styles (e.g., [1, 35]) but without the same quality and generality that can come from learning from examples. Our method allows fine-grained stylistic control as we focus on separating outline and shading style, and learning several stylistic options for each.

(a) Input (b) CycleGAN [48]

(c) Gatys *et al*. [10] (d) Ours: $L_1 + S_2$

(e) Ours: $L_1 + S_4$ (f) Ours: $L_2 + S_3$

Figure 2. Synthesis results of our algorithm in different combinations of outline and shading styles, compared with existing methods (**zoom in** for fine pencil strokes). See Section 4.1 for experimental details.

The main contributions of this work are summarized as follows:

- We propose a two-branch framework that learns one model for generating sketchy outlines and one for tonal shading, from a pencil drawing dataset.
- We show how to use abstraction procedures to generate paired training data for learning to draw with pencils.
- We demonstrate the ability to synthesize images in various different pencil drawing styles within a single framework.
- We present an architecture that captures hatching texture well in the shading drawing, unlike existing baselines.

2. Related Work

Procedural line drawing. There is a rich literature on procedural (non-learning) stylization in Non-Photorealistic Rendering (NPR) [35, 1]. Early work focuses on interactive pen-and-ink drawing and hatching of 2D inputs [36, 37] and 3D models [22, 45, 46]. Pencil drawing is similar to pen-and-ink drawing, but it has more degrees-of-freedom since individual pencil strokes may have varying tone, width, and texture. For 2D images, several procedural image stylization approaches have simulated pencil drawings [24, 34]. These methods use hand-crafted algorithms and features for outlines and a pre-defined set of pencil texture examples for shading. While procedural approaches can be fast and interpretable, accurately capturing a wide range of illustration styles with purely procedural methods is still challenging.

Image-to-image translation. Due to the difficulty in authoring stylization algorithms, numerous approaches have been proposed to learn them from examples. The Image Analogy approach uses texture synthesis applied to a single training pair [15], requiring strict alignment between the input photograph and the output drawing or painting. For line drawing, the paired input images are created manually by applying blurring and sharpening operators separately to each input drawing. This method performs well only for restricted classes of drawings, e.g., when most hatching strokes have a consistent orientation. Although neural image-translation methods have been recently developed [18, 49, 43], none of these has been demonstrated for stylization, due to the difficulty of gathering aligned, paired training data. Chen *et al*. [5] do learn stylization, but trained on the results generated by an existing procedural pencil rendering method [32]. However, the rendered drawings exhibit limited sketching and shading styles. More recently, several methods have been developed for learning mappings from unpaired data [48, 23, 31, 17, 25] with two conditional adversarial losses and cycle-consistency regularization. However, as shown in Figure 2(b), these methods do not perform well at capturing hatching texture. Although the Deep Image Analogy [30] method does not require paired data, it requires data where the exemplar and target photo have very similar content. Several learning-based algorithms have been developed solely for facial portraiture [44, 41, 2, 9].

Neural style transfer. A third approach is to transfer deep texture statistics of a style exemplar, which does not employ paired training data. Since Gatys *et al*. [10] proposed an algorithm for artistic stylization based on matching the correlations (Gram matrix) between deep features, numerous methods have been developed for improvements in different aspects [20], e.g., efficiency [21, 42], generality [5, 16, 28, 4, 27], quality [26, 30, 19], diversity [27], high-resolution [38], and photorealism [33, 29]. However, these methods do not perform well for pencil drawing. The rendered results (Figure 2(c)) in the pencil style only capture the overall gray tones, but without capturing distinctive hatching or outline styles well.

Style control. Procedural methods often provide fine-grained style control, e.g., [13], but at the cost of considerable effort and difficulty in mastering certain styles. Image-to-image translation [17, 25] and neural style transfer meth-

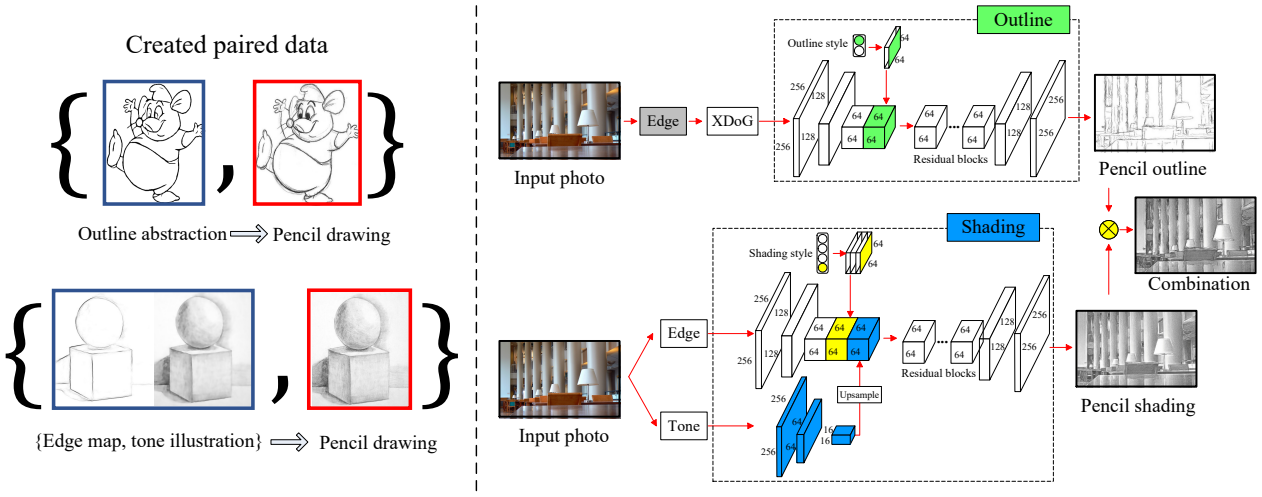


Figure 3. **Pipeline of the proposed algorithm**. Left: the created paired training data generated by using an abstraction procedure on pencil drawings for training. Right: the testing phase (including network details). Two branches will output an outline and shading drawing result respectively, which can be combined together through pixel-wise multiplication as the third option of pencil drawing result. The edge module in gray in the outline branch (top) is a boundary detector [7], which is optional at test-time. For highly-textured photos, it is suggested to use this module to detect boundaries only. See Section 3 for technical details.

ods provide only high-level control, e.g., by selecting training inputs in a different style, interpolating between unrelated styles [8, 11], or selecting among high-level transfer parameters [11]. In this work, we focus on developing a method with fine-grained style control that allows subtle adjustments to pencil drawing.

3. Stylization Approach

Our approach is based on the observation that pencil drawings can be separated into two components: outlines, and shading. The outlines delineate object boundaries and other boundaries in the scene, and shading or tone uses tonal techniques such as hatching to depict reflected lighting, texture, and materials. Hence, our method includes a separate outline branch and a shading branch. These two models are trained separately but can be combined at test-time to generate different combinations of illustration styles. Figure 3 shows the main modules of the proposed method.

For each network branch, paired training data is unavailable, and thus we need to create the input-output pairs from line drawings directly. We generate training data by using an abstraction procedure on pencil drawings, where the abstraction estimates outlines or edges and tones. These filters are designed to produce similar abstractions from line drawings as from photographs. Hence, at test-time, the same abstraction filters can be applied to an input photograph, to produce an input in the same domain as the training inputs.

3.1. Outline branch

The goal of the outline branch is to produce pencil-like outlines from photos. Since there is no paired training data for this task, we use an outline extraction algorithm, both to process the training data and test images at run-time.

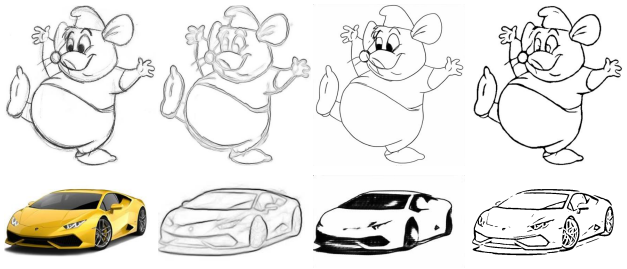
Outline extraction. We use the Extended Difference-of-Gaussians (XDoG) filter [47], which performs well whether the input photo is a pencil drawing or a photograph. The XDoG method takes an input image I, and convolves it with two separate Gaussian filters, with standard deviations σ and $k \cdot \sigma$. A sigmoidal function is then applied to the difference of these two images:

$$D(I;\sigma,k,\tau) = G_\sigma(I) - \tau \cdot G_{k\cdot\sigma}(I), \tag{1}$$

$$E_X(D,\epsilon,\varphi) = \begin{cases} 1, \text{ if } D \geq \epsilon \\ 1 + \tanh(\varphi \cdot (D-\epsilon)), \text{otherwise} \end{cases} \tag{2}$$

The behavior of the filter is determined by five parameters: $\{\sigma, k, \tau, \epsilon, \varphi\}$ for flexible control over detected edges. At test-time, users can adjust any parameters to control the line thickness and sketchiness as shown in Figure 10 and 11.

To demonstrate the effectiveness of XDoG, we compare it with two alternative approaches for outline abstraction. The first one is a boundary detector based on structured random forests [7]. The results in Figure 4(b) show that although it detects outlines in photos well, it generally does not handle thick lines in pencil drawings well and generates two strokes. The second one is a method designed specifically for sketch simplification [39]. As shown on the top of Figure 4(c), it obtains simplification results on main contours but does not handle smooth non-outline regions well (e.g., eyes). More importantly, this sketch simplification method does not perform well on abstracting outlines of



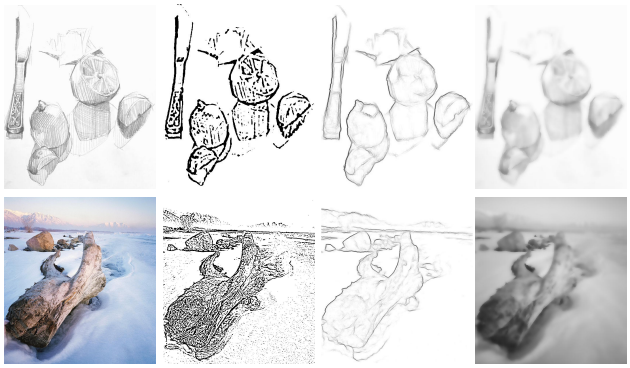
(a) Input (b) [7] (c) [39] (d) XDoG [47]

Figure 4. Comparisons of different outline abstraction results on pencil drawings (top) and photos (bottom).

photo inputs (see the bottom row of Figure 4(c)). In contrast, the XDoG filter handles line thickness and smooth non-outline regions well (Figure 4(d)). For some highly-textured photos at test-time, the XDoG may produce far too many edges (the second row of Figure 5(b)). In these cases, we first use the boundary detector [7] to extract their contours, which are then filtered by the XDoG.

Paired training data. In order to generate paired training data, we first gather a set of pencil *outline* drawings with very little shading, from online websites (e.g., Pinterest). We annotate each drawing with one of two outline style labels: "rough" or "clean" (Figure 1(a)). The data is collected by searching the outline style as the main query on web. We collected 30 images for each style. We use a 2-bit one-hot vector as a selection unit to represent these two styles, which serves as another network input to guide the generation towards the selected style. Then for each drawing, we manually select a set of XDoG parameters that produce good outlines. For example, for a sketchier input, we use a bigger σ to produce one single thick line to cover all sketchy lines along the same boundary. We crop patches of size 256×256 on the created paired data and conduct various augmentations (e.g., rotation, shift), resulting in about 1200 training pairs.

Translation model. As shown on the top row of Figure 3(right), the translation model is designed as an auto-encoder with a few residual-based convolutional blocks in-between. The selection unit is first mapped from a 2-bit vector to a 2-channel map, which is then encoded as feature maps (through convolutions) and concatenated with the features of the outline input. Before the translation module, an XDoG filter is used to extract outlines from photos. This module is *not* included during training, and the rest of the model is trained on the outline/drawing pairs described above. For highly-textured images, the boundary (edge) detector may optionally be used prior to XDoG as well. As described in Section 4.2, adjusting parameters to this XDoG filter can be used to vary outline drawing styles, such as line thickness and sketchiness.



(a) Input (b) XDoG [47] (c) Edge [7] (d) Tone

Figure 5. Examples of extracted edge and tone results for highly-textured inputs.

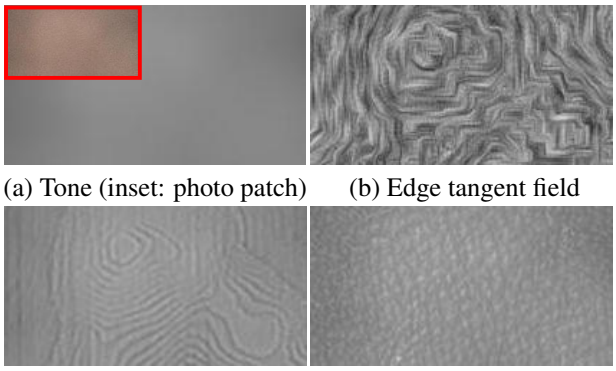
3.2. Shading branch

The goal of our shading branch is to generate textures in non-outline regions according to the tonal values of the input. As no paired data is available for learning the shading branch network, we apply an abstraction procedure to generate the training data and preprocess inputs at test-time. The abstraction aims at extracting edges and tones, and removing detailed textures. Then the model learns to apply pencil shading according to these maps.

Edge and tone extraction. For the edge map, we use the boundary detector by Dollár and Zitnick [7], which identities important edges, even in highly-textured images. We do not use XDoG after boundary detection, because clean outlines are not necessary for shading generation. An example comparison between XDoG and the boundary detector for a highly-textured input is shown in Figure 5(b) and (c).

To extract the tone map, we apply the Guided Filter (GF) [14] on the luminance channel of shading drawings or photos to remove details and generate a smoothing output as the tone extraction. Examples of extracted tone results are shown in Figure 5(d).

Paired training data. We collect a set of pencil *shading* drawings from online websites, and annotate each drawing with one of our four style labels, i.e., *hatching*, *crosshatching*, *blending*, and *stippling* (Figure 1(b)). We searched the data with each shading style as the main web query and collected 20 shading drawings for each style. As in the outline branch, we use a 4-bit one-hot vector as a selection unit. For each drawing, we extract its edge map and tone map to construct the paired data. We manually select the best parameters (e.g., the neighborhood size in GF [14]) for each shading drawing to produce good abstraction results. By cropping patches of size 256×256 and doing data augmentations (e.g., rotation) on the paired data, we create about 3000 training pairs.



(a) Tone (inset: photo patch) (b) Edge tangent field

(c) Single stream tone→shading (d) Two-stream {edge, tone}→ shading

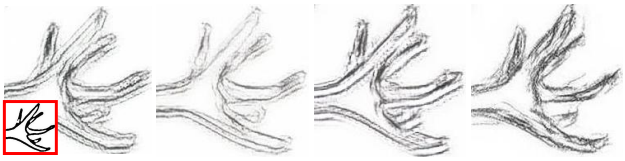
Figure 6. Comparisons of shading results on a photo patch (c)-(d) obtained using different network architectures. The input is a smooth photo patch (red inset). (a) Extracted tone map. (b) Edge tangent field of (a). (c) Hatching result from single-stream architecture. Artificial patterns appear that are unlike normal hatching. (d) Hatching result from two-stream architecture. Hatching-like textures are produced and artificial patterns are suppressed.

Translation model. We find that the direct translation from the extracted tone to the shading drawing generates significant artifacts when applied to photos, especially in smooth regions. In such regions, artists draw pencil textures with varying orientations; these strokes typically approximate the tone but not the specific gradients in the image. However, naive training produces results that attempt to follow the input gradients too closely. Figure 6(a) shows a smooth photo patch and its extracted tone. We visualize its gradient field in terms of edge tangent field which is perpendicular to the image gradients using the linear integral convolutions [3]. When simply relying on the tonal input, the shading result in Figure 6(c) shows that the generated hatching lines looks quite unnatural by following these small gradients.

To address the above-mentioned issue, we design a two-stream translation model to generate the shading (Figure 3(right-bottom)). The main stream is from the edge map of the input, where there is no indication of small image gradients. We employ the tonal abstraction for weak guidance of tone in a secondary input stream that is fused into the main stream at a deeper layer. Figure 6(d) shows that the shading output of our two-stream network architecture significantly reduces the artifacts and exhibits more natural and realistic strokes. In addition, the 4-bit selection unit is fed to the network in the same way as in the outline branch to guide the generation towards different shading styles.

3.3. Learning to draw

With the paired data, we train the model to learn to translate from abstracted inputs to drawings. Our training is based on the existing translation frameworks, e.g., Pix2Pix [18]. However, we use different loss functions for pencil drawing as the existing ones do not perform well for our task. We now describe our loss functions used for both branches of our model.



(a) L_{rec}+L_{adv} (b) L_{per}+L_{adv} (c) L_{rec}+$\sum L^i_{adv}$ (d) Ours

Figure 7. Comparisons of pencil outline results obtained by models trained with different loss functions. Inset image in red rectangle: the XDoG input. L_{rec}: reconstruction loss. L_{adv}: adversarial loss using single discriminator on patches of 256×256. L_{per}: perceptual loss. $\sum L^i_{adv}$: adversarial loss using three discriminators on patches of 256×256, 128×128 and 64×64. Our final loss is the combination of L_{per} and $\sum L^i_{adv}$.

Perceptual loss. We use the perceptual loss [21] to minimize the difference between the network output and the ground truth (GT) pencil drawing based on their deep features:

$$L_{per} = \sum_{i=1}^{4} || \Phi_i(G(x)) - \Phi_i(y) ||_2^2 \ , \tag{3}$$

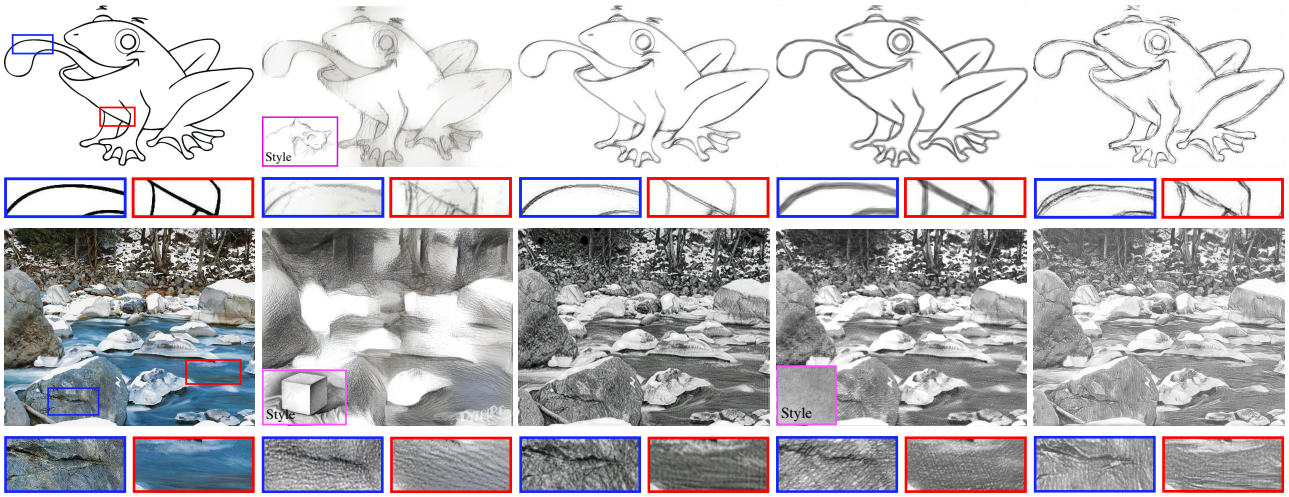
where x, y are the input and the GT pencil drawing, G is the translation model, and Φ_i is the VGG-19 [40] network up to the ReLU_i_1 layer. The feature-based perceptual loss has been shown to generate sharper results than the pixel-based reconstruction loss L_{rec}.

Adversarial loss. In addition to the translation model G, we use the discriminator network D to discriminate between real samples from the pencil drawings and generated results. The goal of G is to generate images that cannot be distinguished by D. This can be achieved by using an adversarial loss [12]:

$$\begin{aligned} L_{adv} = \min_G \max_D \ & \mathcal{E}_{y\sim P_Y}[\log D(y)] \\ & + \mathcal{E}_{x\sim P_X}[\log(1 - D(G(x)))], \end{aligned} \tag{4}$$

where P_Y and P_X represent the distributions of pencil drawing samples y and their abstracted samples x.

To better capture both the global style and local strokes of pencil drawings, we propose to discriminate between real data and fake generations at *multiple scales* during the training. Specifically, given the generated output of size 256×256, we use three discriminators to discriminate patches on three scales (256×256, 128×128, 64×64). Each discriminator is designed as the PatchGAN used in [18].



(a) Input (b) Gatys *et al*. [10] (c) CycleGAN [48] (d) Lu *et al*. [32] (e) Ours

Figure 8. Visual comparisons of different methods for rendering pencil drawing effects (**zoom in for details**). The exemplars used for [10, 32] are shown in the pink rectangle. Top: to directly show pencil effects for the outline, we select a simple line input which is only filtered by the XDoG (no need to detect boundaries first). Bottom: pencil shading effects on a real photo example.

The overall loss function is defined by:

$$L = L_{per} + \beta \sum_{i=1}^{3} L_{adv}^{i}, \tag{5}$$

where β is the weight to balance different losses, and L_{adv}^{i} is the adversarial loss of the ith discriminator. We set $\beta = 100$ in all experiments. When learning with multiple styles, in each iteration, all examples in the batch are limited to be of the same style. Meanwhile, we set the corresponding bit that represents the selected style as 1 and leave other bits as 0 in the selection unit.

Figure 7 shows the outline results from models trained with different losses. It is observed that the perceptual loss encourages better sharpness for a single line drawn along the silhouette compared with the reconstruction loss. Meanwhile, using multiple discriminators helps synthesize better pencil strokes with more sketchy details than employing just one discriminator. Figure 7(d) shows that outline results obtained by the proposed method using the loss function in (5) look more like a real drawing.

4. Experimental Results

In this section, we present extensive experimental results to demonstrate the effectiveness of our algorithm. We compare with methods from both NPR and deep neural network-based stylization. We experiment with synthesizing pencil drawings in various outline and shading styles in a user-controllable manner. More results and comparisons are shown in the supplementary material[1].

[1] http://bit.ly/cvpr19-im2pencil-supp

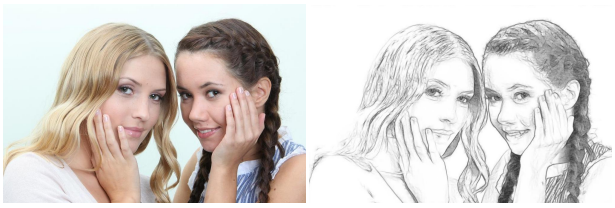
Table 1. User preference towards different methods (%). Each row represents one user study, comparing three stylization algorithms. The top row is applied to the input image directly, and the bottom row uses the tonal adjustment of [32] as a preprocess.

Methods	CycleGAN [48]	Lu *et al*. [32]	Ours
Original tone	10.3	11.4	**78.3**
Adjusted tone	7.1	32.6	**60.3**

4.1. Comparisons

We compare with three algorithms [10, 48, 32] that represent neural style transfer, unpaired image-to-image translation, and NPR respectively. As the method of Gatys *et al*. [10] is example-based, we select a representative pencil outline example (the pink inset in Figure 8(b)) from our dataset to obtain their style transfer results. For CycleGAN [48], in order to train a model for pencil drawing, we collect a photo dataset that consists of 100 images from online websites. Together with our pencil dataset, they construct an unpaired dataset that is used to train CycleGAN for translation between the two domains. Note that since CycleGAN only supports transferring a certain kind of style, one needs to train different CycleGAN model for each outline and shading styles. The NPR method of Lu *et al*. [32] has a two-phase design as well, treating the outline and shading drawing separately. The shading drawing phase also requires a real pencil shading example. Since Lu *et al*. [32] do not release the shading example used for their results, we select a representative pencil shading example (the pink inset in Figure 8(d)) from our dataset to generate their results.

Figure 8 shows the visual comparison of three meth-



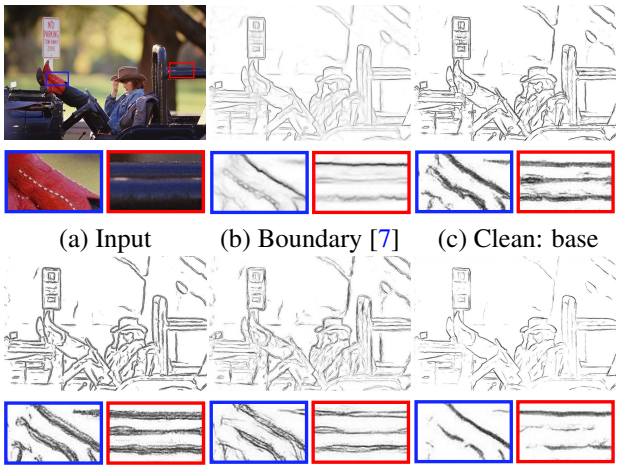
(a) Input (b) [32] w/ tone adjust

(c) Ours w/o tone adjust (d) Ours w/ tone adjust

Figure 9. Comparisons of shading results between w/ and w/o adjusting the tone of input.

ods on outline drawing (top) and shading drawing (bottom). The results by Gatys *et al*. [10] in (b) only exhibit some global gray-like feel of pencil drawing. The lines are unnatural and shading is not correctly positioned to reflect the contrast in the input. CycleGAN [48] generates a few random textures around outlines and inside regions, leading to results that look like the gray image of the input without clear pencil strokes, as shown in Figure 8(c). Without a paired correspondence, simply relying on the cycle constraint and adversarial training is still limited in capturing the real distribution of the pencil drawing domain, with realistic strokes. The shading results of Lu *et al*. [32] in the second row of Figure 8(d) show shading lines and crossings that have no correlation with the underlying lines, structures and "natural" orientations of some content (e.g., the water flow). The shading lines come from a real drawing but the overall result does not look like one. In addition, their gradient-based features also result in detecting the two sides of thick lines in the first row of Figure 8(d), which is uncommon in drawing strokes. In contrast, our results in Figure 8(e) present more realistic pencil strokes in reasonable drawing directions and contain better shading that corresponds to the contrast in the input.

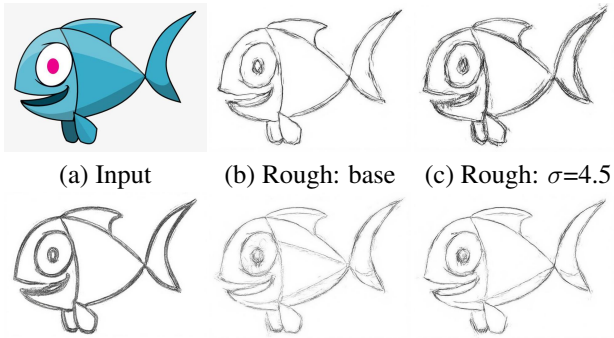
User study. We resort to user studies for the quantitative evaluation of [48, 32] and our method as pencil drawing synthesis is originally a highly subjective task. The method of Gatys *et al*. [10] is not included in the user study because it was clearly inferior to the others in our early experiments (Figure 8(b)). We use 30 natural images provided in [32] and randomly select 15 images for each subject. We display the results by all three methods side-by-side in random order and ask each subject to vote for one result that looks the most like a pencil drawing. We finally collect the feedback from 50 subjects of totally 750 votes and show the



(a) Input (b) Boundary [7] (c) Clean: base

(d) Clean: σ=3 (e) Rough: σ=3 (f) Clean: τ=0.97

Figure 10. Outline results for a highly-textured photo. (b) is the boundary map of input (a), which is then filtered by the XDoG with different parameters. We set $\sigma = 2.0$,$\tau = 0.99$, $k = 1.6$, $\epsilon = 0.1$, $\varphi = 200$ in XDoG and show the base pencil outlines in (c). (d)-(f) show the results by adjusting one parameter while keeping others fixed.



(a) Input (b) Rough: base (c) Rough: σ=4.5

(d) Clean: σ=4.5 (e) Rough: τ=0.98 (f) Rough: ϵ=1.6

Figure 11. Outline results for a simple cartoon image. The input in (a) is directly filtered by the XDoG. We set $\sigma = 2.5$,$\tau = 0.96$, $k = 1.6$, $\epsilon = 0.1$, $\varphi = 200$ and show the base pencil outlines in (b). (c)-(f) show diverse outline results by controlling parameters.

percentage of votes each method received in the top row of Table 1. The study shows that our method receives the most votes for better pencil effects, nearly seven times as much as those of other two methods.

Lu *et al*. [32] observed that pencil drawings often exhibit global tonal changes from the input, and described a histogram-based tone adjustment step to model this observation. In order to fairly compare with this step, we perform a second user study where the input is preprocessed by this step. The user study results with tone adjustment are shown in the bottom row of Table 1. Again, our method obtains substantially more votes than the previous methods. We show our results with and without tone adjustment in

Figure 12. Four types of shading results of the proposed algorithm by switching bits in the selection unit (**zoom in** for details).

Figure 9(c) and (d) as well as the corresponding result of Lu *et al*. [32] in (b). The tone adjustment step provides an additional user control for our method as well.

4.2. User control

Our translation model provides fine-grained control over different pencil drawing styles. Figure 10 and 11 show that users could either switch between clean and rough outline style through the selection unit, or adjust parameters in XDoG to obtain different outline results. The photo example in Figure 10(a) is highly-textured in the clothes and background, so we first use the boundary detector [7] to detect its boundary map, which is then filtered by the XDoG. As the most sensitive and important parameter in XDoG, the σ defines the line thickness and sketchiness. Generally, when a clean style is selected, increasing the value of σ leads to thicker lines (Figure 10(c)-(d)). When a rough style is selected, increasing the value of σ results in increase in repetitive lines (Figure 10(e)). In addition, by adjusting other parameters (e.g., τ), the XDoG filter is able to control the sensitivity on detected edges, which allows users to draw both strong and weak edges (Figure 10(f)). In Figure 11, we show the outline results for a simple cartoon image without heavy textures. Figure 12 shows shading results of two examples, i.e., a still life and a portrait – two popular reference choices for a pencil drawing. By controlling the selection unit, users get results in different shading styles.

Color pencil drawings. The extension of our algorithm to color pencil drawing is quite straightforward, following the method of [15, 11]. We first convert the input image from RGB to LAB color space, then replace the L channel with that of our generated gray-scale pencil drawing result, and finally map back to RGB space. Figure 13 shows two color pencil results in different outline and shading styles.

(a) Input (b) Ours: $L_1 + S_2$ (c) Ours: $L_2 + S_4$

Figure 13. Extension of our algorithm to color pencils in different outline and shading styles (**zoom in** for fine details). Pencil outlines of all examples are generated by applying the XDoG and learned model on their boundary maps.

5. Conclusions

In this work, we propose a photo-to-pencil translation method with flexible control over different drawing styles. We design a two-branch network that learns separate filters for outline and shading generation respectively. To facilitate the network training, we introduce filtering/abstraction techniques into deep models that avoid the heavy burden of collecting paired data. Our model enables multi-style synthesis in a single network to produce diverse results. We demonstrate the effectiveness and flexibility of the proposed algorithm on different pencil outline and shading styles.

Acknowledgment. This work is supported in part by the NSF CAREER Grant #1149783, and gifts from Adobe, Verisk, and NEC. YJL is supported by Adobe and Snap Inc. Research Fellowship.

References

[1] P. Bénard and A. Hertzmann. Line drawings from 3d models. *arXiv preprint arXiv:1810.01175*, 2018. 1, 2

[2] I. Berger, A. Shamir, M. Mahler, E. Carter, and J. Hodgins. Style and abstraction in portrait sketching. *ACM Transactions on Graphics*, 32(4):55, 2013. 2

[3] B. Cabral and L. C. Leedom. Imaging vector fields using line integral convolution. In *SIGGRAPH*, 1993. 5

[4] D. Chen, L. Yuan, J. Liao, N. Yu, and G. Hua. Stylebank: An explicit representation for neural image style transfer. In *CVPR*, 2017. 2

[5] Q. Chen, J. Xu, and V. Koltun. Fast image processing with fully-convolutional networks. In *ICCV*, 2017. 2

[6] F. Cole, A. Golovinskiy, A. Limpaecher, H. S. Barros, A. Finkelstein, T. Funkhouser, and S. Rusinkiewicz. Where do people draw lines? *ACM Transactions on Graphics*, 27(3):88, 2008. 1

[7] P. Dollár and C. L. Zitnick. Structured forests for fast edge detection. In *ICCV*, 2013. 3, 4, 7, 8

[8] V. Dumoulin, J. Shlens, and M. Kudlur. A learned representation for artistic style. In *ICLR*, 2017. 3

[9] J. Fišer, O. Jamriška, M. Lukáč, E. Shechtman, P. Asente, J. Lu, and D. Sỳkora. Stylit: illumination-guided example-based stylization of 3d renderings. *ACM Transactions on Graphics*, 35(4):92, 2016. 2

[10] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *CVPR*, 2016. 2, 6, 7

[11] L. A. Gatys, A. S. Ecker, M. Bethge, A. Hertzmann, and E. Shechtman. Controlling perceptual factors in neural style transfer. In *CVPR*, 2017. 3, 8

[12] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NIPS*, 2014. 5

[13] S. Grabli, E. Turquin, F. Durand, and F. X. Sillion. Programmable rendering of line drawing from 3d scenes. *ACM Transactions on Graphics*, 29(2):18, 2010. 2

[14] K. He, J. Sun, and X. Tang. Guided image filtering. *PAMI*, 35(6):1397–1409, 2013. 4

[15] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin. Image analogies. In *SIGGRAPH*, 2001. 2, 8

[16] X. Huang and S. Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *ICCV*, 2017. 2

[17] X. Huang, M.-Y. Liu, S. Belongie, and J. Kautz. Multimodal unsupervised image-to-image translation. In *ECCV*, 2018. 2

[18] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017. 2, 5

[19] Y. Jing, Y. Liu, Y. Yang, Z. Feng, Y. Yu, D. Tao, and M. Song. Stroke controllable fast style transfer with adaptive receptive fields. In *ECCV*, 2018. 2

[20] Y. Jing, Y. Yang, Z. Feng, J. Ye, Y. Yu, and M. Song. Neural style transfer: A review. *arXiv preprint arXiv:1705.04058*, 2017. 2

[21] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, 2016. 2, 5

[22] E. Kalogerakis, D. Nowrouzezahrai, S. Breslav, and A. Hertzmann. Learning hatching for pen-and-ink illustration of surfaces. *ACM Transactions on Graphics*, 31(1):1, 2012. 2

[23] T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim. Learning to discover cross-domain relations with generative adversarial networks. In *ICML*, 2017. 2

[24] H. Lee, S. Kwon, and S. Lee. Real-time pencil rendering. In *NPAR*, 2006. 2

[25] H.-Y. Lee, H.-Y. Tseng, J.-B. Huang, M. Singh, and M.-H. Yang. Diverse image-to-image translation via disentangled representations. In *ECCV*, 2018. 2

[26] C. Li and M. Wand. Combining markov random fields and convolutional neural networks for image synthesis. In *CVPR*, 2016. 2

[27] Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu, and M.-H. Yang. Diversified texture synthesis with feed-forward networks. In *CVPR*, 2017. 2

[28] Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu, and M.-H. Yang. Universal style transfer via feature transforms. In *NIPS*, 2017. 2

[29] Y. Li, M.-Y. Liu, X. Li, M.-H. Yang, and J. Kautz. A closed-form solution to photorealistic image stylization. In *ECCV*, 2018. 2

[30] J. Liao, Y. Yao, L. Yuan, G. Hua, and S. B. Kang. Visual attribute transfer through deep image analogy. *ACM Transactions on Graphics*, 36(4), 2017. 2

[31] M.-Y. Liu, T. Breuel, and J. Kautz. Unsupervised image-to-image translation networks. In *NIPS*, 2017. 2

[32] C. Lu, L. Xu, and J. Jia. Combining sketch and tone for pencil drawing production. In *NPAR*, 2012. 2, 6, 7, 8

[33] F. Luan, S. Paris, E. Shechtman, and K. Bala. Deep photo style transfer. In *CVPR*, 2017. 2

[34] X. Mao. Automatic generation of pencil drawing from 2d images using line integral convolution. In *CAD/GRAPHICS*, 2001. 2

[35] P. Rosin and J. Collomosse. *Image and Video-Based Artistic Stylisation*. Springer, 2013. 1, 2

[36] M. P. Salisbury, S. E. Anderson, R. Barzel, and D. H. Salesin. Interactive pen-and-ink illustration. In *SIGGRAPH*, 1994. 2

[37] M. P. Salisbury, M. T. Wong, J. F. Hughes, and D. H. Salesin. Orientable textures for image-based pen-and-ink illustration. In *SIGGRAPH*, 1997. 2

[38] A. Sanakoyeu, D. Kotovenko, S. Lang, and B. Ommer. A style-aware content loss for real-time hd style transfer. In *ECCV*, 2018. 2

[39] E. Simo-Serra, S. Iizuka, K. Sasaki, and H. Ishikawa. Learning to simplify: Fully convolutional networks for rough sketch cleanup. *ACM Transactions on Graphics*, 35(4), 2016. 3, 4

[40] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 5

[41] Y. Song, L. Bao, Q. Yang, and M.-H. Yang. Real-time exemplar-based face sketch synthesis. In *ECCV*, 2014. 2

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

[43] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *CVPR*, 2018. 2

[44] X. Wang and X. Tang. Face photo-sketch synthesis and recognition. *PAMI*, 31(11):1955–1967, 2009. 2

[45] G. Winkenbach and D. H. Salesin. Computer-generated pen-and-ink illustration. In *SIGGRAPH*, 1994. 2

[46] G. Winkenbach and D. H. Salesin. Rendering parametric surfaces in pen and ink. In *SIGGRAPH*, 1996. 2

[47] H. Winnemöller, J. E. Kyprianidis, and S. C. Olsen. XDoG: an extended difference-of-gaussians compendium including advanced image stylization. *Computers & Graphics*, 36(6):740–753, 2012. 3, 4

[48] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *ICCV*, 2017. 2, 6, 7

[49] J.-Y. Zhu, R. Zhang, D. Pathak, T. Darrell, A. A. Efros, O. Wang, and E. Shechtman. Toward multimodal image-to-image translation. In *NIPS*, 2017. 2