

Image Stylization: From Predefined to Personalized

Ignacio Garcia-Dorado, Pascal Getreuer, Bartłomiej Wronski, Peyman Milanfar

Google Research

Abstract:

We present a framework for interactive design of new image stylizations using a wide range of predefined filter blocks. Both novel and off-the-shelf image filtering and rendering techniques are extended and combined to allow the user to unleash their creativity to intuitively invent, modify, and tune new styles from a given set of filters. In parallel to this manual design, we propose a novel procedural approach that automatically assembles sequences of filters, leading to unique and novel styles. An important aim of our framework is to allow for interactive exploration and design, as well as to enable videos and camera streams to be stylized on the fly. In order to achieve this real-time performance, we use the *Best Linear Adaptive Enhancement* (BLADE) framework – an interpretable shallow machine learning method that simulates complex filter blocks in real time. Our representative results include over a dozen styles designed using our interactive tool, a set of styles created procedurally, and new filters trained with our BLADE approach.

1 Introduction

From the moment the camera was invented there has always been an interest to raise the bar of realism: capturing higher resolution images, inclusion, improvement, and precision of color, or even the addition of 3D scene depth. Fine art and photography have different goals when it comes to rendering a scene. While the former focuses on the aesthetic where the artist reflects their creative ideas using brush, paint, and a blank canvas, the latter aims to capture the intent of the photographer using a technical piece of equipment and (in more modern incarnations) software. *Stylization* of photos is an approach that bridges the two worlds, allowing us to explore artistic expression on a canvas of already-captured images. Stylization creates evocative, abstract representations of natural and synthetic scenes, and is not limited to static photographs. It can also be applied to on video footage and video games as an additional dimension of artistic expression [1].

Related to stylization is the concept of *rotoscoping*, an animation technique used to trace over motion pictures footage, frame by frame, to produce realistic action. The technology dates back to the 1910s when Max Fleischer used it for cinematic storytelling. Besides the usage of classic cartoons like *Popeye* and *Betty Boop*, it garnered special attention with the '80s music video *Take on Me* [2]. A few additional examples have been created since then, including *A Scanner Darkly* [3] in the '90s, and the recent masterpiece *Loving Vincent* [1]. This technique allows the creation of stunning visual effects that enable the animator to dramatically change the style. The main drawback is the manual labor.

In the early 90's, stylization was used by artists and designers to communicate, to abstract their ideas, and to express themselves. Technical illustration uses stylization to explain better the object's parts. By using the silhouettes and feature lines of the object it is easier to communicate concepts and remove spurious detail. The first example of stylization is arguably Haeberli [4]. This seminal work achieved stylization by creating painterly images from a collection of brush strokes that were computed using local attributes of an image. This work was extended in the research community by many others ([5–9]). Automated stylization enabled practical application of stylization to video, for the first time used in the movie *What Dreams May Come* [10]. Since then, many movies (e.g., *Waking Life* [11] and *Sim City* [12]) and an increasing number of apps have included stylization of their content.

Nowadays, in the era of mobile photography, widely popular applications such as Instagram, Snapchat, Facebook, and Google Photos use stylization filters to alter the captured world. Millions of pictures are shared daily and in most cases the photos are processed with some filters. Since most pictures are stylized, the need to identify pictures that have *not* been altered has increased. Users often identify such pictures with the hash-tag '#nofilter' to show that no alterations were made (e.g., in Instagram there are over 260M photos posted with such a tag). We use stylization to express our emotions and feelings [13] and to increase the likelihood that our pictures are viewed and engaged with in social media [14]. In this work we propose to extend the expressiveness of a photograph by allowing the user to create their own personalized stylizations. Note that although this kind of stylization can be achieved with specialized tools such as Photoshop and Gimp, the required knowledge is out of reach of most users. Our goal is to bring user-customisable stylization to users of mobile devices, which requires reduced processing cost and simpler UI compared to desktop. Our work will describe both a wide set of predefined stylizations, to the creation of unique stylizations tuned and designed by the user.

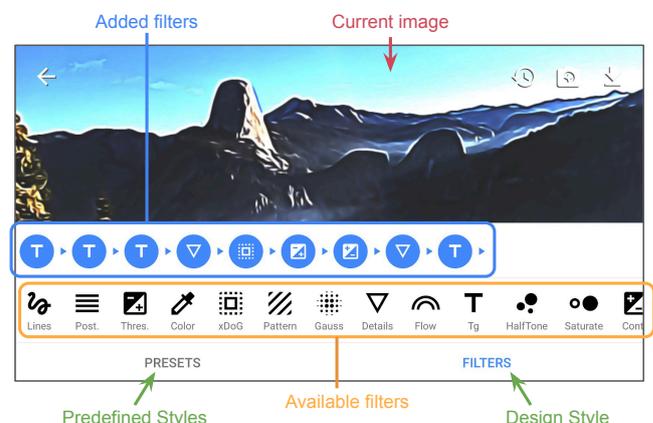


Fig. 1 Experimental Design App. The users adds/modifies/replaces over 20 block filters to create a new stylization using a real time design application. A set of predefined styles are available as a starting point to be further customized by the user. See Sec. 4.1 for details.

We present an interactive design framework (see Fig. 1) to create new stylizations using a wide range of predefined filter blocks. While the most common filters can be off-the-shelf image processing and rendering techniques, here we apply and combine them in novel ways. When designing the filter blocks (see Sec. 4.2), we recognize that the most interesting filters (e.g., Flow-XDoG) are too slow for interactive or real-time use. We expand the *Best Linear Adaptive Enhancement* (BLADE) of Getreuer et al. [15], a lightweight, yet effective learning approach that is *trainable* yet *computationally simple* and *interpretable*, to emulate a chosen set of filters in this stylization task. We show that using a shallow and easy to train machine learning method, we can add to our design toolbox a set of accurate approximations of significantly more complex and expensive filters. Our approach can be seen as a shallow two-layer network, where the first layer is predetermined and the second layer is trained. We show that this simple network structure allows for inference that is computationally efficient, easy to train, interpretable, and sufficiently flexible to perform the task of complex stylization. The main contributions of this work can be summarized as follows:

- We present an interactive design framework for stylization. The resulting tool allows for modifying, tuning, and designing new styles on the fly.
- We extend BLADE, a light-weight and interpretable machine learning framework that is straightforward to train, to able to perform real-time inference of complex filters on mobile devices.
- We propose a novel procedural style generation technique built on our design tool. Through simple rules we create original stylization effects from novel and automated combinations of filter blocks.

The rest of the paper is organized as follows: Section 2 reviews previous work. Section 3 is an overview of our system. Section 4 introduces our design application and the filter blocks. Section 5 explains how we use BLADE to achieve real time performance. Section 6 presents our stylization design results. Finally, Section 7 contains conclusions and future work.

2 Related work

We cover related work in filtering and stylization research. Note that stylization of images and videos represents a broad research area, thus we review three main topics: filtering, video stylization, style transfer, and learnable filtering.

2.1 Filtering and Abstraction

Winnemöller et al. [9] proposed the use of eXtended Difference-of-Gaussians (XDoG) to create interesting sketch and hatching effects. Kang et al. [6] extended XDoG by adding an edge tangent flow block to create smooth edges.

Kyprianidis and Döllner [7] used oriented separable filters and XDoG to achieve a high level of image abstraction. Kang et al. [8] further improved the level of abstraction by adding a flow-based step. Other more complex algorithms simplify images using advanced multi-scale detail image decomposition [16].

For more general background on modern approaches to image filtering, we refer the reader to [17, 18]. For a more comprehensive survey of artistic stylizations we refer the reader to Kyprianidis et al. [19]. We employ some of these filters as building blocks of our stylization system as described in Section 4.

2.2 Video Stylization

As an extension of image filtering, some works have focused on speeding up the stylization process to run at interactive rates. Winnemöller et al. [20] used a bilateral filter iteratively to abstract the input, then quantize the background color and overlay XDoG to produce strong outlines. Our system can achieve similar results using a different set of filters. Barnes et al. [21] proposed to precompute a multidimensional hash table to accelerate the process of finding

replacement patches. This structure enables them to stylize a video in real time using a large collection of patch examples

2.3 Style Transfer

As an alternative to explicit filter creation, a wide range of works have developed a technique called *style-transfer*. Style transfer is a process of migrating a style from a given image (reference) to the content of another (target), synthesizing a new image which is an aesthetic mixture of the two. Recent work on this problem uses Convolutional Neural Networks (CNN). Gatys et al. [22] posed the style-transfer problem as an energy minimization task, seeking an image close to the target using output of a CNN as a loss function. As an alternative to CNNs, Elad and Milanfar [23] presented an approach based on texture synthesis. Their approach copies patches from the reference image to the target while maintaining the main features of the content image using a hierarchical structure.

Despite the visual appeal of these approaches, their complexity is a major drawback. The method described by Gatys et al. [24] can take up to an hour to stylize a single image. More recent works have focused on addressing this issue. Johnson et al. [25] achieved real-time style transfer with simplified networks running on a high-end desktop GPU. However, achieving similar results on a full HD image on a mobile device would require tens of seconds. Elad and Milanfar's [23] approach takes minutes to run on a mobile device.

Another alternative is patch-based style transfer. Such methods transfer the style by finding and applying patches of the reference image in the target image. Barnes et al. [21] presented a method to efficiently query patches within a large dataset and replace each patch of the target image with one from the reference image. Friego et al. [26] used local image features to determine the best scale of a patch. These approaches do not allow control over the color, line weight, and other features of stylization.

Style transfer has several other drawbacks. First, output quality depends directly on the reference image. While often considered an advantage, having a specific template can generate inconsistent and undesirable results for different inputs (e.g., very bright/dark images) or transform the content in areas in which we would like to preserve details. Second, current style transfer approaches do not provide sufficient aesthetic control. Gatys et al. [24] extended their own work to introduce control over color, scale, and spatial location. However, this does not allow the fine tuning required to design new stylizations.

Note that our method does not compete against these approaches; each of the techniques above can be incorporated into our system as a new block to further enrich the creative toolbox. Moreover, note that our method focuses on the design aspect of new stylizations, and while we aim to create real-time filters that can create interesting results, we also want to allow for creating filters that can be tuned to a specific designer intent.

2.4 Trainable Filters

As previously discussed, we may express many methods of stylization as a cascade of filtering operations which are adaptive to image content in some way. Turning the problem on its head, we are able to discover and parameterise a sequence of filters which emulate the effects of those more complicated systems. These *trainable filter* approaches include the recent Deep Learning approaches, variational methods, or closely connected methods in partial differential equations, Markov random fields, and maximum a posteriori estimation. A particularly successful direction is “unrolling” (or “unfolding”), which is described generically for instance by Liu et al. [27]. The recipe is to formulate a task as an optimization, solve it with an iterative algorithm (e.g. with gradient descent or proximal methods), unroll several iterations, then substitute portions of the algorithm with trainable parameters. Compared to generic convolutional architectures, the advantage of this unrolling approach is that it tends to reduce the needed number of parameters, training data, and inference computational cost for a given level of quality. For instance Chen and Pock's trainable nonlinear reaction

diffusion [28] and Lefkimmiatis’s work [29] are image denoising networks designed by unrolling variational optimization algorithms. Besides denoising, deep unrolling has been applied to tasks such as image deblurring [27, 30, 31]. While deep learning methods are capable of achieving impressive quality, they are hard to analyze and debug, and still too expensive to run interactively on smartphones at full-resolution.

For mobile-friendly filtering, we extend the Best Linear Adaptive Enhancement (BLADE) framework of Getreuer et al. [15] (based on the RAISR method of Romano et al. [32]) to achieve real time inference of complex filters. For instance with 5×5 filters on the Google Pixel 2018 phone, our CPU implementation runs at 38.21 MP/s and our GPU implementation at 223.03 MP/s.

3 Overview

In this section, we show an overview of our work. Fig. 2 illustrates our two workflows for stylization design. In the first workflow (top) the user provides as input an image or uses a video live-stream (e.g., selfie camera stream) and manually selects filters and modifies parameters until the desired effect is achieved. In Sec. 4 we present the interactive interface (Sec. 4.1) and the filter blocks (Sec. 4.2). In the second workflow (bottom) a style sequence is automatically generated by evaluating random combinations of filters and parameter sets. We call this *procedural stylization*. In this case, style sequences are selected based on manual or automated subjective assessment (NIMA [33], a no-reference aesthetic prediction network).

In addition to relatively well-known linear filters (e.g., blurring and saturation) we include more sophisticated effects based on the use of BLADE filters (Fig. 3) to enable real time inference on mobile devices. Sec. 5 explains the details of BLADE: inference (5.1), training (5.2), filter selection with structure tensor features (5.3), and real-time filtering (5.4).

4 Stylization through filtering

In this section, we describe an interactive tool we created for designing styles by combining filter blocks.

4.1 Interactive Style Design

While most works (Sec. 2.1) focus on creating one filter or a fixed set of filters to achieve a style, our goal is to create a flexible tool

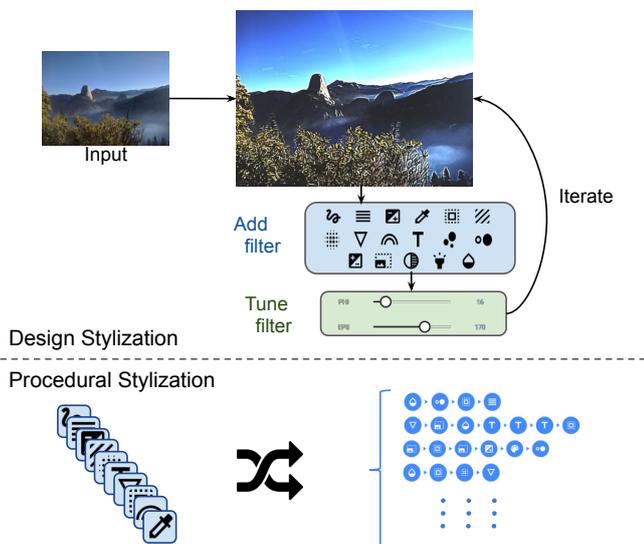


Fig. 2 Diagram of our method: Stylization design and procedural generation.

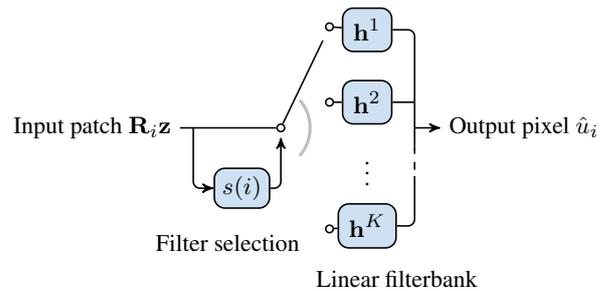


Fig. 3 BLADE inference. \mathbf{R}_i denotes extraction of a patch centered at pixel i . For a given output pixel \hat{u}_i , we only need to evaluate the one linear filter that is selected, $h^{s(i)}$.

that allows anyone to design stylization filters, regardless of their technical ability.

The final result of our framework is as shown in Fig. 1. The main components are:

1. A wide range of filter blocks (see Sec. 4.2).
2. The ability to tune the parameters for any filter through sliders.
3. Two layers: a black and white foreground layer used as alpha channel to display lines and contours; and a background layer for color. This separation provides added flexibility.
4. A visual flow diagram of filter blocks which allow any filter to be added, moved, reordered, removed, and tuned at any time. This is the essence of the system’s interaction design.

4.2 Filter blocks

We implemented three categories of filter blocks: pixel operations, advanced filters, and histogram modification filters. The effect of each filter block on an example image is shown in Fig. 4.

A. Pixel-wise Operators:

- **To Grayscale:** Converts the image into grayscale/luminance and saves the chrominance channels (UV). This block is useful for applying any other block filter to just the luminance channel. [Fig. 4 (b)].
- **To Color:** Uses the current luma and converts the image back to RGB using the UV previously saved by the *To Grayscale* block.
- **Posterization:** Discretizes continuous image colors (e.g., 255 levels) to regions of fewer tones, e.g. $levels = 10$, [Fig. 4 (c)].
- **Luma Posterization:** Posterizes the image in the luma channel by converting the image to grayscale, applying *Posterization*, and converting it back to color [Fig. 4 (d)].
- **Brightness:** Multiplies the luma channel by a user-selected *brightness* constant, clipping the output values [Fig. 4 (e)].
- **Soft Threshold [9]:** Computes the following function for each pixel

$$output = 1 + \tanh(\min(0, \phi \cdot (input - \epsilon))) \quad (1)$$

where ϕ determines the slope and ϵ the cut-off. For grayscale images, this block behaves like a binary cut-off that preserves smooth transitions. For color images, it simplifies each RGB channel into two levels [Fig. 4 (f)].

- **Saturation:** Makes the colors more vivid or more muted by adding or subtracting in RGB the grayscale image tuned by a parameter [Fig. 4 (g)].
- **Hue:** Performs a color rotation in UV space and adds a bias in RGB . This block is useful for changing the image’s tint [Fig. 4 (h)].
- **Colorize:** Convert to monochrome using an HSL palette transformation [Fig. 4 (i)].

B. Spatial Filters:



Fig. 4 Filters: (a) Input, (b) To Grayscale, (c) Posterization, (d) Luma Posterization, (e) Brightness, (f) Soft Threshold, (g) Saturation, (h) Hue, (i) Colorize, (j) Gaussian Smoothing, (k) ETF, (l) TVF, (m) Sobel, (n) XDoG, (o) Pattern Filling, (p) Halftone, (q) Image Detail Control, and (r) Linear Equalization. See text for details.

- **Gaussian Smoothing:** Blurs and removes details and noise from the image with a controllable standard deviation (σ) parameter [Fig. 4 (j)].
- **Sobel Filter [34]:** Fast edge-detection filter [Fig. 4 (m)].
- **Scale:** Upscales or downscales the image using a user-selected scale parameter. This block can help to speed up computation (computing other blocks in a lower resolution) and alter the behavior of other scale-dependent filters.
- **Pattern Filling Filter:** Uses *Luma Posterization* to discretize the image into a set of levels, then, each pixel is replaced by a texture depending on its level. This block is useful for creating cross-hatching patterns [Fig. 4 (o)].
- **Halftone:** Replaces colors with a set of dots that vary in size and color. This style mimics the behavior of the four-color printing process traditionally used to print comics [Fig. 4 (p)].

C. BLADE Filters:

- **Edge Tangent Flow (ETF) [6]:** Creates an impressionistic oil painting effect. This method uses a kernel-based nonlinear smoothing of vector field inspired by bilateral filtering [Fig. 4 (k)].
- **Total Variation Flow (TVF) [35]:** Makes image piecewise constant with an anisotropic diffusion filter [Fig. 4 (l)].
- **Flow XDoG [9]:** Uses *Difference-of-Gaussians* to find the edges of the image. In our implementation, the user can control the variance of the main Gaussian (σ) and the multiplier (p) [Fig. 4 (n)].
- **Detail Control [16]:** Controls the details of the image by adding the residual of the image to its filtered version multiplied by a δ . Setting $\delta < 0$ smooths the image while $\delta > 0$ adds details [Fig. 4 (q)].

D. Histogram Operators:

- **Linear Histogram Equalization:** Equalizes the luma channel expanding the p_l to zero and the p_h to 255. Normally we choose $l = 5$ and $h = 95$ such that the 5% percentile is moved to zero and the 95% is moved to 255, thereby increasing the image's dynamic range [Fig. 4 (r)].
- **Histogram Minimum Dynamic Range:** Computes the percentile 5% and 95% on the luma histogram and expands (if necessary) the dynamic range to match the user parameter range DR .

These two filters are usually placed as the first filter in the pipeline to force the image to have a proper dynamic range to obtain a satisfactory result. For instance applying XDoG on a hazy or too bright/dark image would result in an almost entirely white output image.

5 Best Linear Adaptive Enhancement

As we mentioned, the trade-off between quality and speed is a challenge with stylization. We want the freedom to apply elaborate techniques to produce interesting style effects, but on the other hand, computational efficiency is necessary to run efficiently. This is especially the case when processing full-resolution images or video as part of an interactive application on a mobile device. To this end, we leverage the BLADE framework to learn fast approximations to more complex filters.

Let \mathbf{z} be an input grayscale image and \mathbf{u} the target output grayscale image. We denote by subscript z_i the i th pixel value at spatial position $i \in \Omega \subset \mathbb{Z}^2$. Let $\mathbf{h}^1, \dots, \mathbf{h}^K$ denote a set of K linear FIR filters, each having footprint or nonzero support $F \subset \mathbb{Z}^2$. The coefficients of these filters are learned.

5.1 Inference

We describe inference first to introduce the structure of the BLADE network architecture. BLADE inference is a spatially-varying filter. For each output pixel, one filter in the bank is selected and applied:

$$\hat{u}_i = \sum_{j \in F} h_j^{s(i)} z_{i+j}, \quad (2)$$

where $s(i) \in \{1, \dots, K\}$ denotes the index of the filter selected at the i th pixel (see filter selection at Sec. 5.3). Equivalently, inference (2) can be written in vector notation as

$$\hat{u}_i = (\mathbf{h}^{s(i)})^T \mathbf{R}_i \mathbf{z}, \quad (3)$$

where $(\cdot)^T$ denotes matrix transpose and \mathbf{R}_i is the patch extraction operator defined by $(\mathbf{R}_i \mathbf{z})_j := z_{i+j}, j \in F$.

When computing \hat{u}_i , only the selected filter needs to be evaluated. The complexity per pixel is $O(N)$ where $N = |F|$ is the number of pixels in the footprint. Notably, computation cost is independent of the number of filters K . Furthermore, inference is independent for each output pixel, so it is readily parallelized and implemented with high efficiency.

We use features of the 2×2 image structure tensor for the filter selection $s(i)$. This makes BLADE filtering adaptive to image edges and structure. In principle, BLADE can operate with any deterministic function of \mathbf{z} as the selection rule.

To run BLADE on color images, we use the *To Grayscale* and *To Color* operations described earlier to extract the luma channel, filter luma with (2), and reassemble a color image.

5.2 Training

To train the BLADE filters $\mathbf{h}^1, \dots, \mathbf{h}^K$ to approximate an existing style effect, we first apply a (possibly slow) reference implementation of the effect to a set of images to create a training set of input image / target output image example pairs. We also train on 90° rotations and flips of the images, augmenting the training set by a factor of 8.

For notational simplicity, we describe training for a single example image pair \mathbf{z} and \mathbf{u} . The filters $\mathbf{h}^1, \dots, \mathbf{h}^K$ are trained with a simple L^2 loss plus a quadratic regularization term,

$$\arg \min_{\mathbf{h}^1, \dots, \mathbf{h}^K} \sum_{k=1}^K \left((\mathbf{h}^k)^T \mathbf{Q} \mathbf{h}^k + \sum_{\substack{i \in \Omega: \\ s(i)=k}} |u_i - (\mathbf{h}^k)^T \mathbf{R}_i \mathbf{z}|^2 \right). \quad (4)$$

The matrix \mathbf{Q} determines the regularization. To encourage spatially-smooth filters, we set it to a discretization of L^2 norm of the filter's spatial gradient ([15]). The inner sum is over the set of pixels i where the k th filter is selected.

The training minimization (4) decouples over the filters, so each filter can be solved independently.

For each filter \mathbf{h} , its solution amounts to a multivariate linear regression with regularization. Denote by $\{i(1), \dots, i(M)\}$ an enumeration of pixels where $s(i) = k$, and M the number of such pixels. The optimal filter \mathbf{h}^k is

$$\mathbf{h} = (\mathbf{Q} + \mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{b} \quad (5)$$

where $A_{m,n} = (\mathbf{R}_{i(m)} \mathbf{z})_n$ and $b_m = \mathbf{u}_{i(m)}$. Rather than storing \mathbf{A} and \mathbf{b} themselves, it is possible to accumulate $\mathbf{A}^T \mathbf{A}$ as a matrix of size $N \times N$ and $\mathbf{A}^T \mathbf{b}$ as a length- N vector. This way filters can be trained from any number of examples with a fixed amount of memory.

Once the filters $\mathbf{h}^1, \dots, \mathbf{h}^K$ are trained, we visualize them as a tabular collage (Figs. S3, S4, S5, S6). This is often an illuminating characterization of how BLADE will behave, as inference amounts to selecting among and applying these filters. For instance, a successful choice of filter selection mechanism $s(i)$ should allow the filters to specialize and take on diverse shapes, which we can inspect for visually. Additionally as described in detail in [15], the variance of the regression residual is a good diagnostic to identify filters in need of more training examples or stronger regularization.

5.3 Structure tensor features

Following [32] and [15], we make BLADE adaptive to the local image content by basing the filter selection $s(i)$ on features of the 2×2 image structure tensor [36–38]. In continuous space, the structure tensor is a 2×2 matrix of spatial derivatives at every location:

$$J(\nabla u) := \begin{pmatrix} \partial_{x_1} u \\ \partial_{x_2} u \end{pmatrix} \begin{pmatrix} \partial_{x_1} u & \partial_{x_2} u \end{pmatrix}, \quad (6)$$

where above, $u(x)$ is a differentiable continuous-domain image, ∂_{x_1} and ∂_{x_2} denote partial derivatives, and $\nabla = (\partial_{x_1}, \partial_{x_2})^T$ denotes

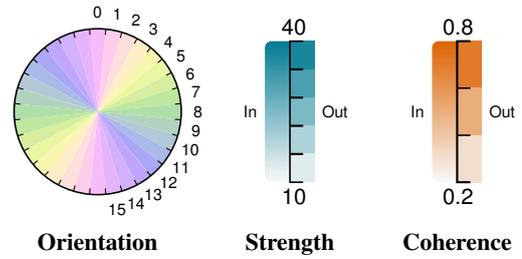


Fig. 5 Quantization of filter selection features with 16 orientation bins, 5 strength bins, and 3 coherence bins.

gradient. In implementation the derivatives can be discretized with finite differences. We consider it important that the two spatial components of the gradient estimate are aligned with one another in order to avoid asymmetrical behavior in the processing. Using the smallest stencil with this property, we compute finite differences in 45° rotated coordinates x'_1, x'_2 over a 2×2 stencil as

$$\frac{1}{\sqrt{2}h} (u(x_1 + 1, x_2) - u(x_1, x_2 + 1)) \quad (7)$$

$$= \partial_{x'_1} u(x_1 + \frac{1}{2}, x_2 + \frac{1}{2}) + O(h^2),$$

$$\frac{1}{\sqrt{2}h} (u(x_1 + 1, x_2 + 1) - u(x_1, x_2)) \quad (8)$$

$$= \partial_{x'_2} u(x_1 + \frac{1}{2}, x_2 + \frac{1}{2}) + O(h^2).$$

Next, each component of the structure tensor is spatially filtered with a Gaussian kernel G_ρ with standard deviation ρ ,

$$J_\rho(\nabla u) := G_\rho * J(\nabla u). \quad (9)$$

The filtered structure tensor $J_\rho(\nabla u)$ is at each location an aggregate of the image statistics of its neighborhood, with the neighborhood size determined by ρ . These statistics robustly capture the pre-dominant edge orientation and other local image characteristics, as studied for instance by Weickert [39], Zhu and Milanfar [40], and Takeda et al. [41].

At the i th pixel, we compute three features from the eigensystem of the 2×2 filtered matrix $J_\rho(\nabla u)_i$: (1) **orientation** as the angle of the dominant eigenvector, (2) **strength** as the square root of the dominant eigenvalue, and (3) **coherence** according to

$$\text{coherence} = \frac{\sqrt{\lambda_1} - \sqrt{\lambda_2}}{\sqrt{\lambda_1} + \sqrt{\lambda_2}} \quad (10)$$

where $\lambda_1 \geq \lambda_2 \geq 0$ are the eigenvalues.

To perform filter selection $s(i)$, we quantize these features to uniform bins, and we then view the bin indices as a three-dimensional index into the bank of filters. Fig. 5 illustrates a typical quantization.

5.4 Real-Time Advanced Filtering

The following sections describe how we take advanced, computationally demanding effects and apply visually accurate approximations of them in real time using the BLADE framework.

5.4.1 TV flow: Total variation (TV) flow is a classic anisotropic diffusion process that tends to regularize the image into piecewise-constant regions. TV flow flattens textures and details while retaining strong edges, which is aesthetically interesting in designing style effects for image simplification and producing a cartoon-like look.

In order to create a BLADE approximation of TV flow, we use as training target the modified TV flow definition of Marquina and Osher [42],

$$\partial_t u = |\nabla u| \operatorname{div}(\nabla u / |\nabla u|), \quad (11)$$

in which $u(x)$ is a continuous-domain image. Compared to the usual TV flow, this definition has an extra $|\nabla u|$ factor, which mitigates the tendency to produce artificial edges in smooth gradients.

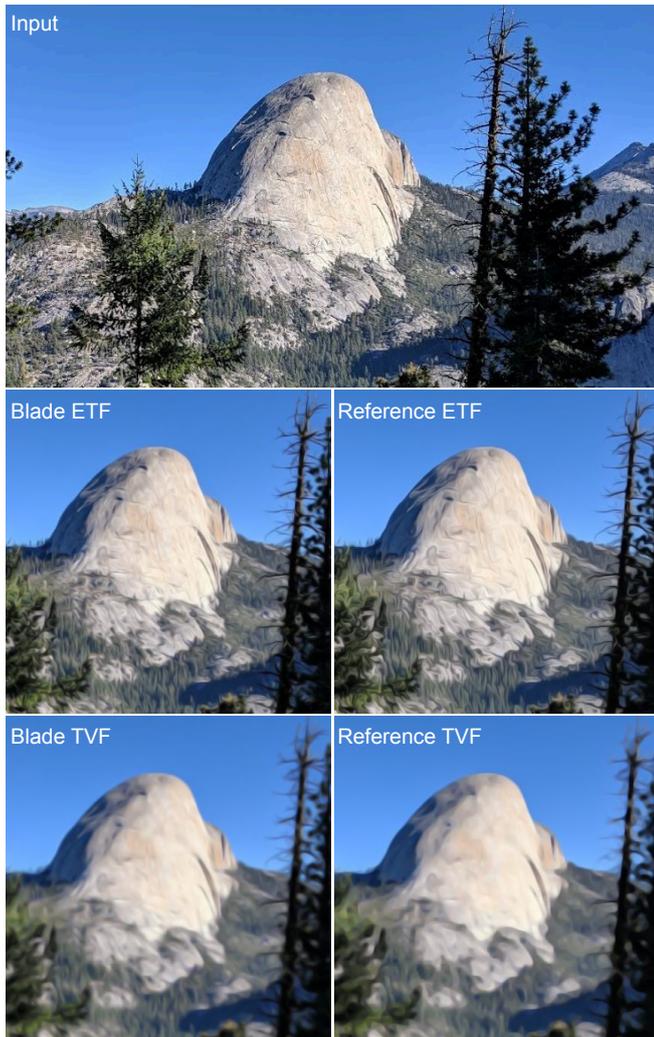


Fig. 6 Example of approximated edge tangent flow and TV flow. BLADE has PSNR 34.85 dB and MSSIM 0.9619 for ETF and PSNR 35.86 dB and 0.9683 for TVF.



Fig. 7 Example of approximated Flow-XDoG. Readers are encouraged to zoom in aggressively (200% or more).

For training, we use a small private dataset of 32 photographs of size 1600×1200 as input images. As a note, we find that the choice of input images does not have an impact, since BLADE does not have enough parameters (e.g., a 5×5 BLADE filter has 5400 parameters) to overfit the training data. To create the training target images, we apply the second-order scheme developed in [42] as the reference implementation to generate TV flow target images. For filter selection, we use 16 orientations, 4 strength bins, and 4 coherence bins ($K = 256$ filters total) and use filters of size 7×7 (Fig. S5).



Fig. 8 Examples of three scales of trained Flow-XDoG. Readers are encouraged to zoom in aggressively (300% or more).

Fig. 6 shows an example, comparing the input, TV flow reference implementation, and the fast BLADE approximation.

5.4.2 Edge tangent flow: Edge tangent flow (ETF) is another anisotropic diffusion equation with useful qualities for style effects. This process smooths the image along edges but not across them, as its name suggests. When applied to a photographic image, ETF tends to alter textures into flows and swirls like an impressionist-style oil painting. When applied as post processing to an edge mask (from the Sobel or Flow XDoG filter blocks), ETF makes lines visually more organic and flowy like hand-drawn strokes.

ETF anisotropic diffusion is defined mathematically by

$$\partial_t u = \text{div}(D(u)\nabla u), \quad (12)$$

where at each spatial location, $D(u)(x)$ is the 2×2 outer product of the unit-magnitude local edge tangent orientation. The edge tangent orientation is obtained as the weaker eigenvector of the smoothed image structure tensor.

For training a BLADE approximation of ETF, we use the same dataset of 32 photographs, and use line integral convolution evolved with second-order Runge–Kutta as a reference implementation to generate ETF target images. We use 24 orientations and 3 coherence bins. We omit the strength feature, since ETF is essentially a one-dimensional diffusion at every point in the edge tangent orientation and this behaves irrespective of the gradient magnitude. We train filters of size 5×5 (Fig. S3). Our CPU implementation on a Xeon E5-1650v3 runs BLADE ETF at 27.1 MP/s. Fig. 6 shows an example.

BLADE ETF has two interesting parameters: increasing the structure tensor smoothing parameter ρ produces broader brush strokes, and applying multiple passes of the filter (more time steps of the diffusion equation) results in longer strokes.

5.4.3 Flow-XDoG: Extracting image edges is a fundamental element for line drawing or cartoon-like styles. We briefly review the origin of the Flow-XDoG filter:

- The classical Marr–Hildreth approach to edge detection is Laplacian filtering, which can be made more noise robust by using a Laplacian of Gaussian filter. The DoG filter $G_{k,\sigma} - G_\sigma$ with $k = 1.6$ is a close approximation to the Laplacian of Gaussian [43].



Fig. 9 Example of approximated Detail Control for values -20 and $+20$. BLADE has PSNR 38.38 dB and MSSIM 0.9845 for -20 and PSNR 41.36 dB and 0.9895 for $+20$.

- Improving on DoG, Kang et al. [6] introduced flow-based difference-of-Gaussians (FDoG). A basic implementation of FDoG is simply ETF followed by DoG filtering, which we could realize as two filter blocks. However, as Kang et al. develop, the edge tangent flow field can be used in a “flow-guided” DoG filter to obtain a cleaner output that responds more strongly on true edges.
- In a later work, Winnemöller et al. [9] introduced extended difference-of Gaussians (XDoG). XDoG substitutes the highpass DoG filter $G_{k\sigma} - G_{\sigma'}$ with a high-emphasis filter $(1+p)G_{k\sigma} - pG_{\sigma'}$, with emphasis according to parameter p . Second, XDoG follows the filter the *Soft Threshold* function (1). XDoG can be combined with FDoG, which we refer to as *Flow-XDoG*.

To make a fast approximation of Flow-XDoG, we decompose it to two stages. First, BLADE is used to approximate a flow-guided version of the high-emphasis DoG filter $(1+p)G_{k\sigma} - pG_{\sigma'}$ (for which we use second-order Runge–Kutta line integral convolution as the reference implementation), and second, the soft threshold is applied. Being a simple pointwise operation, the soft threshold is straightforward to implement separately from BLADE, and this has the advantage that the soft threshold parameters ϕ and ϵ are tunable at inference time.

We use 16 orientations, 5 strength bins, and 3 coherence bins, and we train 7×7 filters (Fig. S4). Fig. 7 shows an example, comparing the input, Flow-XDoG reference implementation, and the fast BLADE approximation. Fig. 8 shows how we can change the

scale of the image at training to produce a more subtle behavior of Flow-XDoG.

5.4.4 Detail Control: Smoothing or enhancing details is a versatile tool for stylization. Smoothing can help to remove the details to create an abstraction or it can enhance the details to emphasize the different aspect of the images. An effective detail control filter is Talebi and Milanfar’s multilayer Laplacian enhancement [16].

For training a BLADE approximation, we use the same dataset of 32 photographs, and use the algorithm described in [16] as a reference implementation to create training targets.

We use 16 orientations, 5 strength bins, and 3 coherence bins, and we train 9×9 filters (Fig. S6). Note this change affects only the luminance channel, therefore Fig. 9 shows an example of control to smooth or increase the detail of that channel.

6 Style Design

In this section, we present the following results: a set of new styles generated by designers and a set of styles generated procedurally.

6.1 Interactive Style Design

We conducted several style design sessions with graphic designers who generated manually dozens of styles. Fig. 10 shows six styles and Fig. 11 shows an additional seven styles to give the reader some appreciation of the breadth of effects that could be deployed with this system.

Fig. S7 depicts the design of each stylization from the input (left) to the final result (right) for six styles and Fig. S2 shows a summary of how the styles are generated.

- **Style 1:** This style uses Flow-XDoG to transform the input then threshold and color are applied. Gaussian block is used to smooth the filtering. Here we present it in an orange tint but we created explored versions with dark blue, green, blue, and grayscale.
- **Style 2:** This style tries to imitate crayons. This stylization uses a smoothed and thresholded version of the image to apply a down-scaled version of Flow-XDoG in grayscale, finally, the colors are saturated.
- **Style 3:** This style tries to create the effect of a sketch. Instead of using XDoG (as it was done in [9]), we simplify this filter by using thresholding and apply a five-level hatching texture to the grayscale. Finally, we overlay lines to highlight contours.
- **Style 4:** This style tries to abstract or simplify the input. To achieve such a result, we remove details, downscale, and smooth with the watery ETF filter, we then apply Flow-XDoG and apply posterization.
- **Style 5:** This is a heavily stylized result. Details are removed and Flow-XDoG is applied and several ETV and TVF are applied. To get the final output, we posterize the result and apply lines to highlight the edges.
- **Style 6:** We call this stylization *blob*, it is a colorful abstraction of the image. To create it, we apply Flow-XDoG in its lowest scale (i.e., it smoothes the input), we posterize the result and remove details.

We show the performance on device and on desktop (Table 1). The table shows that our method, even when using heavy filters (such

Table 1 Style computation time of Fig. 10 and 11 (in megapixels per second) for ‘Device’ (Pixel 2018) and Desktop GPU.

Style	Device MP/s	Desktop MP/s	Style	Device MP/s	Desktop MP/s
1	124.4	1719.4	7	131.0	1637.5
2	156.9	2405.5	8	169.4	2295.9
3	128.4	1469.1	9	114.0	675.2
4	30.3	357.5	10	241.8	4069.9
5	92.3	1469.4	11	240.7	4138.8
6	40.0	458.7	12	126.0	1804.9
			13	101.3	1323.3

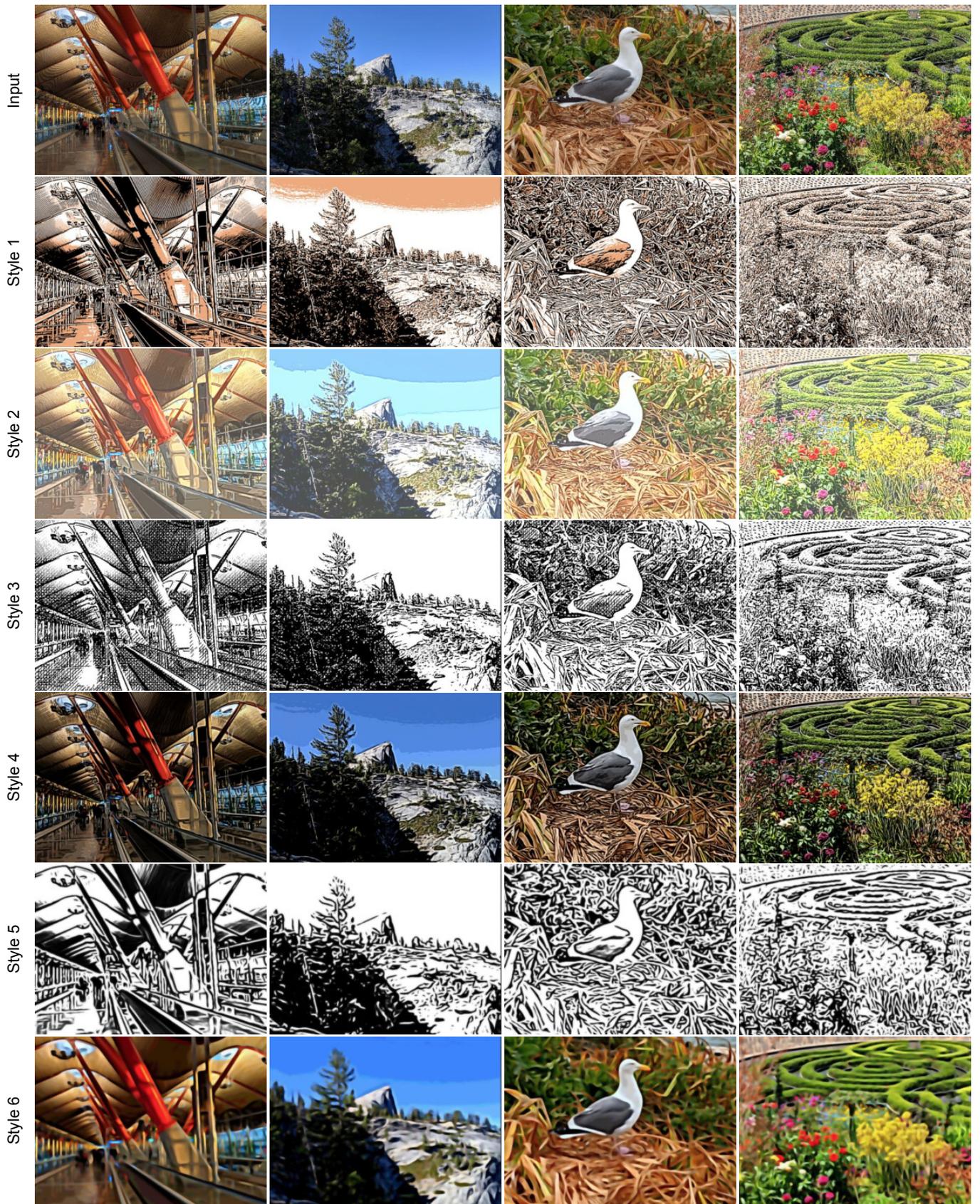


Fig. 10 Input image on top and six different stylizations created with our tool. See details in Sec. 6.1. Readers are encouraged to zoom in aggressively (200% or more).

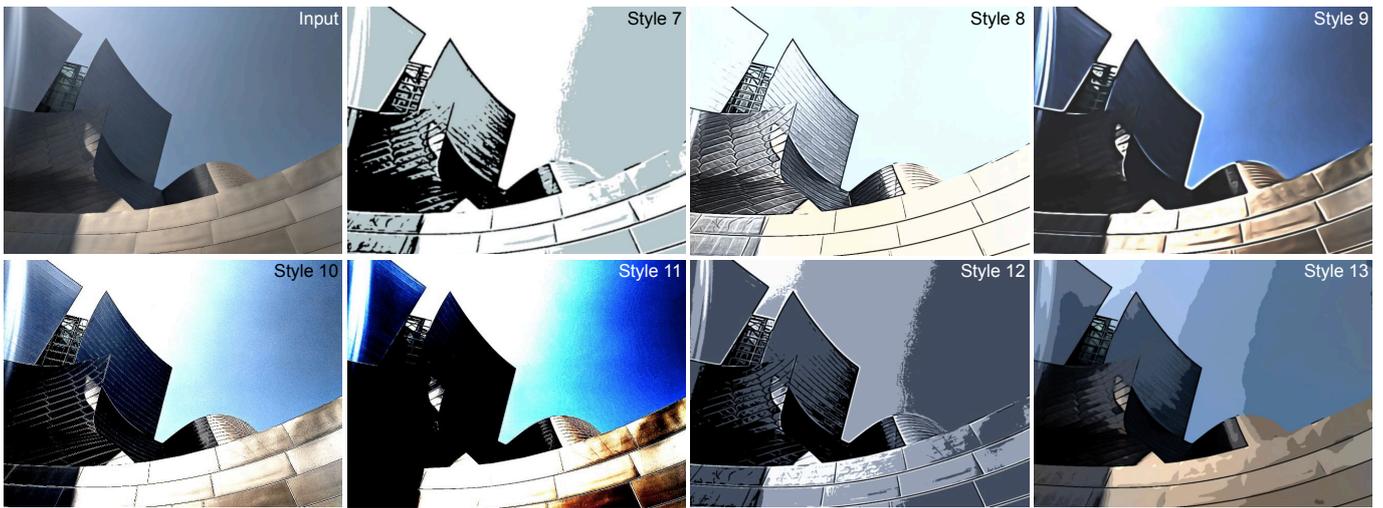


Fig. 11. Additional stylization created with our tool. Readers are encouraged to zoom in aggressively (200% or more).

as three times ETF), can achieve real-time rendering performance. When a style uses the basic set of filters it is possible to process a Full HD input video in under 1ms, when more advanced filters are used we achieve real time rendering over 30fps at viewfinder resolution. Desktop performance is one order of magnitude faster and we can achieve 16K video processing with over 60fps.

6.2 Procedural Styles

Procedural modeling is technique that enables the generation of hundreds or thousands of examples using a limited set of rules. In the computer graphics community, the technique has been used to generate buildings, cities, trees, and more complex models; see for instance [44, 45]. Based on this idea, we call *procedural styles* those created randomly using a set of simple rules.

The simple set of rules for this experiment was to add a random number of filters between 4 and 9, with random input parameters: XDoG ($\sigma \in [0.5, 8.0]$ and $p \in [1, 40]$), TVF, Soft Threshold ($\phi \in [0.013, 0.059]$ and $\epsilon \in [50, 110]$), Detail Control ($\delta \in [-100, 60]$), Luma Posterization (*level* $\in [5, 12]$), Saturation (*saturation* $\in [1.5, 2.2]$), Size (*size* $\in [100, 300]$), and To GrayScale (20% probability). We also enforce a rule that XDoG and TVF are the only filters that can be added more than once (duplicating the rest of filters has the same effect as selecting different parameters).

We developed a web-based visualization tool to quickly review the procedurally generated alternatives (Fig. 12). Fig. 13 shows four examples of styles found manually in under five minutes using the visualization tool. In order to automate this process, we used the



Fig. 12 Visualization tool to explore alternative procedurally generated styles. Many alternatives (as seen in the figure) are not very interesting or distinct, but the system's real-time speed and created visualization tool makes it fast and easy to explore and identify promising alternatives.

Neural Image Assessment (NIMA) network of Talebi and Milanfar [33], which evaluates the aesthetic quality of an image. Using this approach, we discovered the styles in Fig. 14.

6.3 Computational Performance

One of the key goals of our system is allowing for interactive exploration and design of different styles and options, as well as being able to process videos and camera streams. Therefore, we have aimed to achieve real-time computational performance and decided for a GPU implementation. Most image filters are standard image processing operations, and we have implemented them using OpenGL and GLSL shading language.

In the case of BLADE filtering, we have taken advantage of the fact that it is a fully parallelizable algorithm, and also implemented it using GPU GLSL shading language. We have decoupled two main steps of the algorithm as separate full-image passes: filter bucket hash computation and applying of the selected filter. Both passes run only on a single image channel (luminance) and take advantage of accelerated instructions and OpenGL extensions like *ARB_texture_gather* that allow to process four pixels at the same time. To optimize the bucket index computation arithmetic, we use approximations to transcendental functions where applicable. For example the arctangent for orientation angle computation uses a variation of a well-known quadratic approximation [46].

With the GPU implementation and inherent parallelism of our filters, we have observed order of magnitude speed-up over straightforward CPU implementation. We observe linear performance scaling with the number of processed pixels for all implemented filtering operations and real-time performance for Full HD images on a mobile device (Fig. S1 (bottom)) and up to 40MP images on a desktop PC (Fig. S1 (top)).

7 Conclusions

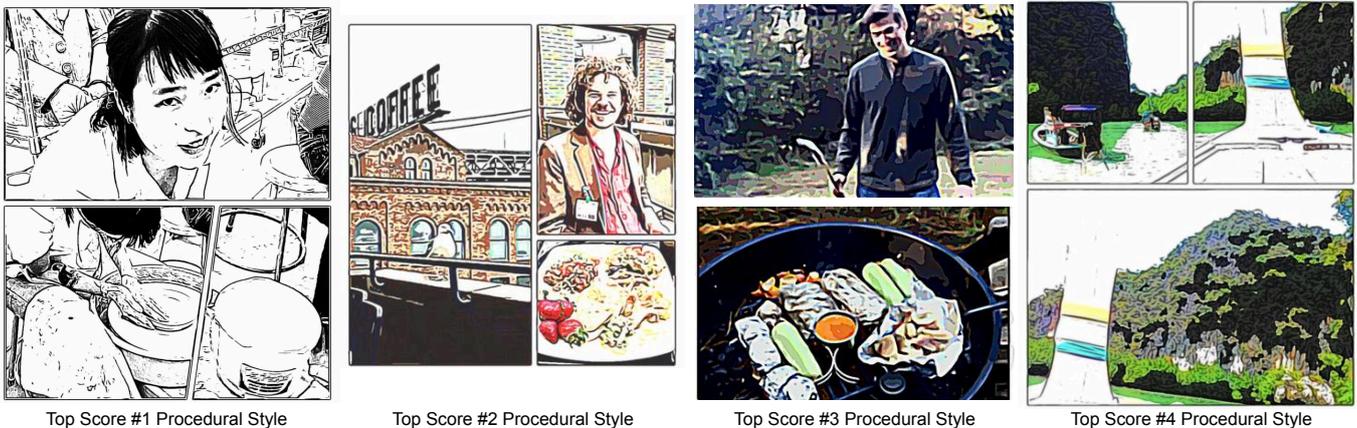
In this work, we presented an interactive framework for designing filter-based stylization which allows a designer to tune, modify, and conveniently explore the space of filters directly, empowering user creativity. Our framework is flexible, and not limited to any particular visual style.

In parallel to this manual design, we presented a procedural, fully automatic style creation process that follows a set of simple rules to generate hundreds of different styles. These styles can be selected through manual visualization or evaluated using a previously trained aesthetics quality assessment method (e.g. [33]). Our filtering and stylization framework was used to design filters



Exploration #1 Procedural Style Exploration #2 Procedural Style Exploration #3 Procedural Style Exploration #4 Procedural Style

Fig. 13. Manual exploration of procedurally generated styles. Readers are encouraged to zoom in aggressively (200% or more).



Top Score #1 Procedural Style Top Score #2 Procedural Style Top Score #3 Procedural Style Top Score #4 Procedural Style

Fig. 14. Top-scored procedurally generated styles. Readers are encouraged to zoom in aggressively (200% or more).

for a free, experimental application called *Google Storyboard** that allows to automatically turn any video into a comic strip.

In order to make the system real-time, we apply the Best Linear Adaptive Enhancement (BLADE) framework for simple, trainable, and edge-adaptive filtering to realize fast approximations of sophisticated style effects. BLADE's computationally efficient inference allows for fully real-time applications on a mobile device and is easy to train and interpret.

For future work, we find several potential directions of explorations. First, one can include any new, more advanced filters. These may include CNN-based stylization approaches that would enrich the expressiveness of our system. A second possibility is to explore more complex hand-crafted features for filter selection. While this might increase the computational requirements, it would allow for more sophisticated and more content-adaptive stylization filters. Finally, the adaptation of the proposed approach to longer videos including episodic-length television and motion pictures, while enabling automatic or convenient change of style as required by the scene or the director, would be quite interesting.

8 References

- Kobiela, D., Welchman, H.: 'Loving Vincent', *Altitude Film Distribution*, 2017,
- Mansfield, T.: 'Take on me', *Warner Bros*, 1984,
- Linklater, R.: 'A Scanner Darkly', *Warner Bros*, 2006,
- Haerberli, P.: ; ACM. 'Paint by numbers: Abstract image representations', *ACM SIGGRAPH*, 1990, **24**, (4), pp. 207–214
- Litwinowicz, P.: ; CiteSeer. 'Processing images and video for an impressionist effect', *Computer Graphics and Interactive Techniques*, 1997, pp. 407–414
- Kang, H., Lee, S., Chui, C.K.: ; ACM. 'Coherent line drawing', *International symposium on Non-photorealistic animation and rendering*, 2007, pp. 43–50
- Kyprianidis, J.E., Döllner, J.: 'Image abstraction by structure adaptive filtering', *TPCG*, 2008, pp. 51–58
- Kang, H., Lee, S., Chui, C.K.: 'Flow-based image abstraction', *IEEE Transactions on Visualization and Computer Graphics*, 2009, **15**, (1), pp. 62–76
- Winnemöller, H., Kyprianidis, J.E., Olsen, S.C.: 'XDoG: an eXtended difference-of-Gaussians compendium including advanced image stylization', *Computers & Graphics*, 2012, **36**, (6), pp. 740–753
- Ward, V.: 'What Dreams May Come', *Universal Studios*, 1998,
- Linklater, R.: 'Waking Life', *Fox Pictures*, 2001,
- Frank.Miller, R.R.: 'Sin City', *Miramax*, 2005,
- Hu, Y., Manikonda, L., Kambhampati, S.: 'What we instagram: A first analysis of instagram photo content and user types', *International AAAI conference on Web and Social Media*, 2014,
- Bakhshi, S., Shamma, D.A., Kennedy, L., Gilbert, E.: 'Why we filter our photos and how it impacts engagement', *International AAAI Conference on Web and Social Media*, 2015,
- Getreuer, P., Garcia-Dorado, I., Isidor, J., Choi, S., Ong, F., Milanfar, P.: ; IEEE. 'BLADE: filter learning for general purpose computational photography', *ICCP*, 2018, pp. 1–11
- Talebi, H., Milanfar, P.: 'Fast multilayer Laplacian enhancement', *IEEE Transactions on Computational Imaging*, 2016, **2**, (4), pp. 496–509
- Milanfar, P.: 'A tour of modern image filtering new insights and methods, both practical and theoretical', *IEEE Signal Processing Magazine*, 2013, **30**, (1), pp. 106–128
- Milanfar, P.: 'Symmetrizing smoothing filters', *SIAM, Journal on Imaging Science*, 2013, **6**, (1), pp. 263–284
- Kyprianidis, J.E., Collomosse, J., Wang, T., Isenberg, T.: 'A taxonomy of artistic stylization techniques for images and video', *IEEE Transactions on Visualization and Computer Graphics*, 2013, **19**, (5), pp. 866–885
- Winnemöller, H., Olsen, S.C., Gooch, B.: 'Real-time video abstraction', *ACM Transactions On Graphics (ToG)*, 2006, **25**, (3), pp. 1221–1226
- Barnes, C., Zhang, F.L., Lou, L., Wu, X., Hu, S.M.: 'Patchtable: Efficient patch queries for large datasets and applications', *ACM Transactions on Graphics (ToG)*, 2015, **34**, (4), pp. 97
- Gatys, L.A., Ecker, A.S., Bethge, M.: 'Image style transfer using convolutional neural networks', *IEEE Conference on Computer Vision and Pattern Recognition*

* <https://play.google.com/store/apps/details?id=com.google.android.apps.photolab.storyboard>

- (CVPR), 2016, pp. 2414–2423
- 23 Elad, M., Milanfar, P.: ‘Style transfer via texture synthesis’, *IEEE Transactions on Image Processing*, 2017, **26**, (5), pp. 2338–2351
 - 24 Gatys, L.A., Ecker, A.S., Bethge, M., Hertzmann, A., Shechtman, E.: ‘Controlling perceptual factors in neural style transfer’, *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 3985–3993
 - 25 Johnson, J., Alahi, A., Fei-Fei, L.: ; Springer. ‘Perceptual losses for real-time style transfer and super-resolution’, *European Conference on Computer Vision*, 2016, pp. 694–711
 - 26 Frigo, O., Sabater, N., Delon, J., Hellier, P.: ‘Split and match: Example-based adaptive patch sampling for unsupervised style transfer’, *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 553–561
 - 27 Liu, R., Fan, X., Cheng, S., Wang, X., Luo, Z.: ‘Proximal alternating direction network: A globally converged deep unrolling framework’, *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
 - 28 Chen, Y., Pock, T.: ‘Trainable nonlinear reaction diffusion: A flexible framework for fast and effective image restoration’, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017, **39**, (6), pp. 1256–1272
 - 29 Lefkimmatis, S.: ‘Universal denoising networks: a novel CNN architecture for image denoising’, *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 3204–3213
 - 30 Corbineau, M.C., Bertocchi, C., Chouzenoux, E., Prato, M., Pesquet, J.C.: ‘Learned image deblurring by unfolding a proximal interior point algorithm’, *IEEE International Conference on Image Processing (ICIP)*, 2019, pp. 4664–4668
 - 31 Li, Y., Tofighi, M., Monga, V., Eldar, Y.C.: ‘An algorithm unrolling approach to deep image deblurring’, *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 7675–7679
 - 32 Romano, Y., Isidoro, J., Milanfar, P.: ‘RAISR: Rapid and Accurate Image Super Resolution’, *IEEE Transactions on Computational Imaging*, 2017, **3**, (1), pp. 110–125
 - 33 Talebi, H., Milanfar, P.: ‘Nima: Neural image assessment’, *IEEE Transactions on Image Processing*, 2018, **27**, (8), pp. 3998–4011
 - 34 Sobel, I.: ‘An isotropic 3×3 image gradient operator’, *Machine vision for three-dimensional scenes*, 1990, pp. 376–379
 - 35 Louchet, C., Moisan, L.: ‘Total variation as a local filter’, *SIAM Journal on Imaging Sciences*, 2011, **4**, (2), pp. 651–694
 - 36 Förstner, W., Gülch, E.: ‘A fast operator for detection and precise location of distinct points, corners and centres of circular features’, *Proceedings of the Intercommission Conference on Fast Processing of Photogrammetric Data*, 1987, pp. 281–305
 - 37 Bigun, J., Granlund, G.H.: ‘Optimal orientation detection of linear symmetry’, *IEEE First International Conference on Computer Vision*, 1987, pp. 433–438
 - 38 Knutsson, H., Westin, C.F.: ‘Normalized and differential convolution’, *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1993, pp. 515–523
 - 39 Weickert, J.: ‘Anisotropic diffusion in image processing’, *Teubner Stuttgart*, 1998.
 - 40 Zhu, X., Milanfar, P.: ‘A no-reference sharpness metric sensitive to blur and noise’, *IEEE International Workshop on Quality of Multimedia Experience*, 2009, pp. 64–69
 - 41 Takeda, H., Farsiu, S., Milanfar, P.: ‘Robust kernel regression for restoration and reconstruction of images from sparse noisy data’, *IEEE International Conference on Image Processing (ICIP)*, 2006.
 - 42 Marquina, A., Osher, S.: ‘Explicit algorithms for a new time dependent model based on level set motion for nonlinear deblurring and noise removal’, *SIAM Journal on Scientific Computing*, 2000, **22**, (2), pp. 387–405
 - 43 Marr, D., Hildreth, E.: ‘Theory of edge detection’, *Proceedings of the Royal Society of London Series B Biological Sciences*, 1980, **207**, (1167), pp. 187–217
 - 44 Müller, P., Wonka, P., Haegler, S., Ulmer, A., Van.Gool, L.: ; ACM. ‘Procedural modeling of buildings’, *ACM Transactions On Graphics (ToG)*, 2006, **25**, (3), pp. 614–623
 - 45 Nishida, G., Garcia.Dorado, I., Aliaga, D.G., Benes, B., Bousseau, A.: ‘Interactive sketching of urban procedural models’, *ACM Transactions on Graphics (ToG)*, 2016, **35**, (4), pp. 130
 - 46 Rajan, S., Wang, S., Inkol, R., Joyal, A.: ‘Efficient approximations for the arctangent function’, *IEEE Signal Processing Magazine*, 2006, pp. 108–111

Appendix

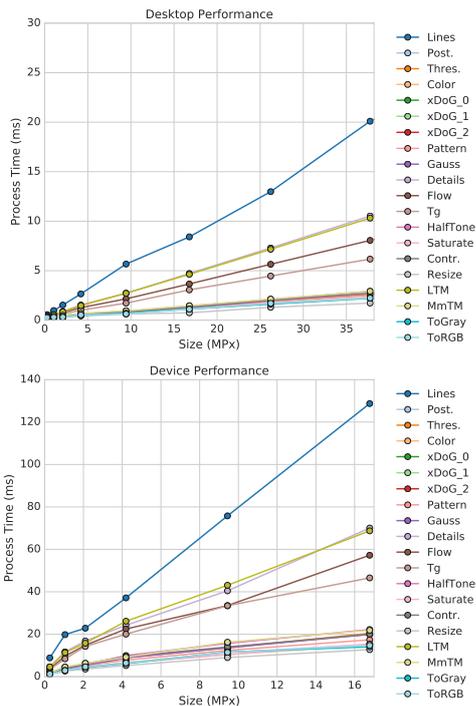


Fig. S1 Process time on GTX 1080 desktop PC and on 'Device' (Pixel 2018) vs. image size in megapixels for each of the filters in our system.

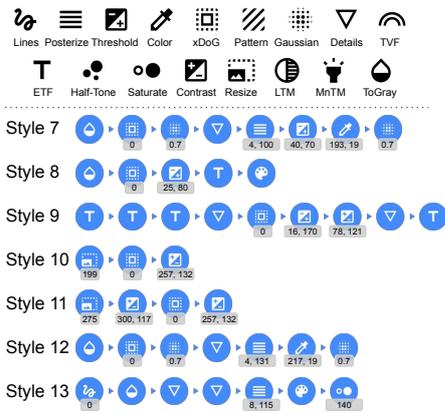


Fig. S2 Graph of each stylization. The caption of each filter are the parameters for the given filter.

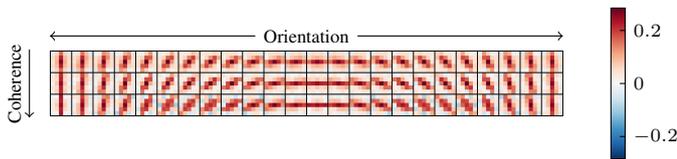


Fig. S3 5 × 5 BLADE filters approximating edge tangent flow with 24 different orientations and 3 coherence values.

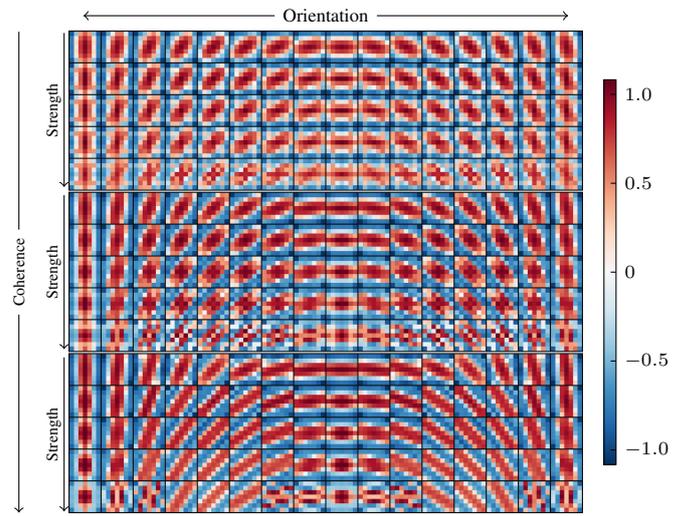


Fig. S4 7 × 7 BLADE filters for approximating Flow-XDoG with 16 different orientations, 5 strength bins, and 3 coherence bins.

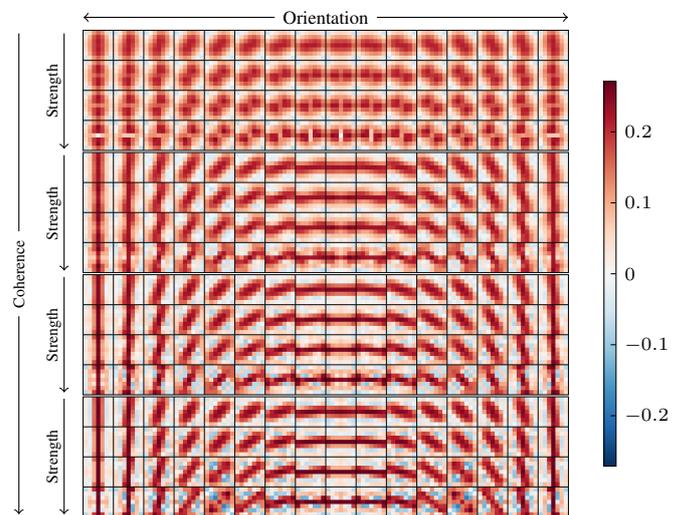


Fig. S5 7 × 7 BLADE filters approximating TV flow with 16 different orientations, 4 strength values, and 4 coherence values.

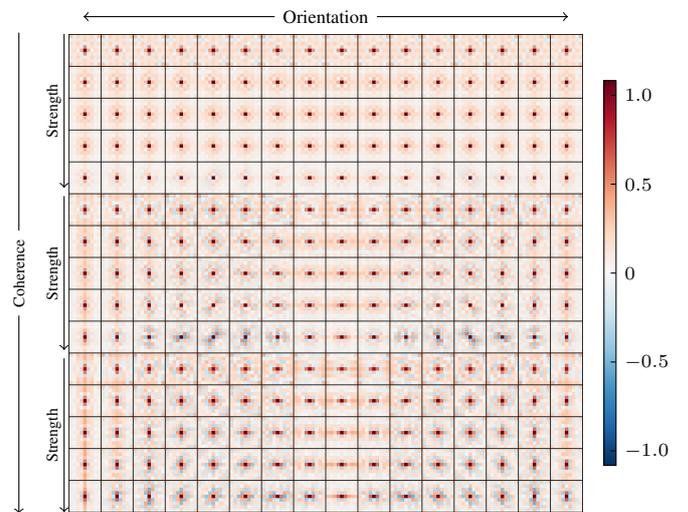


Fig. S6 9 × 9 BLADE filters for approximating Detail Control -20 with 16 different orientations, 5 strength bins, and 3 coherence bins.



Fig. S7 Progression of each stylization from input (left) to the final stylization (right). The top right corner shows the filters applied in that step (see meaning of icons on Fig. S2). Note that Style 3 has just four filters. Readers are encouraged to zoom in aggressively (300% or more).

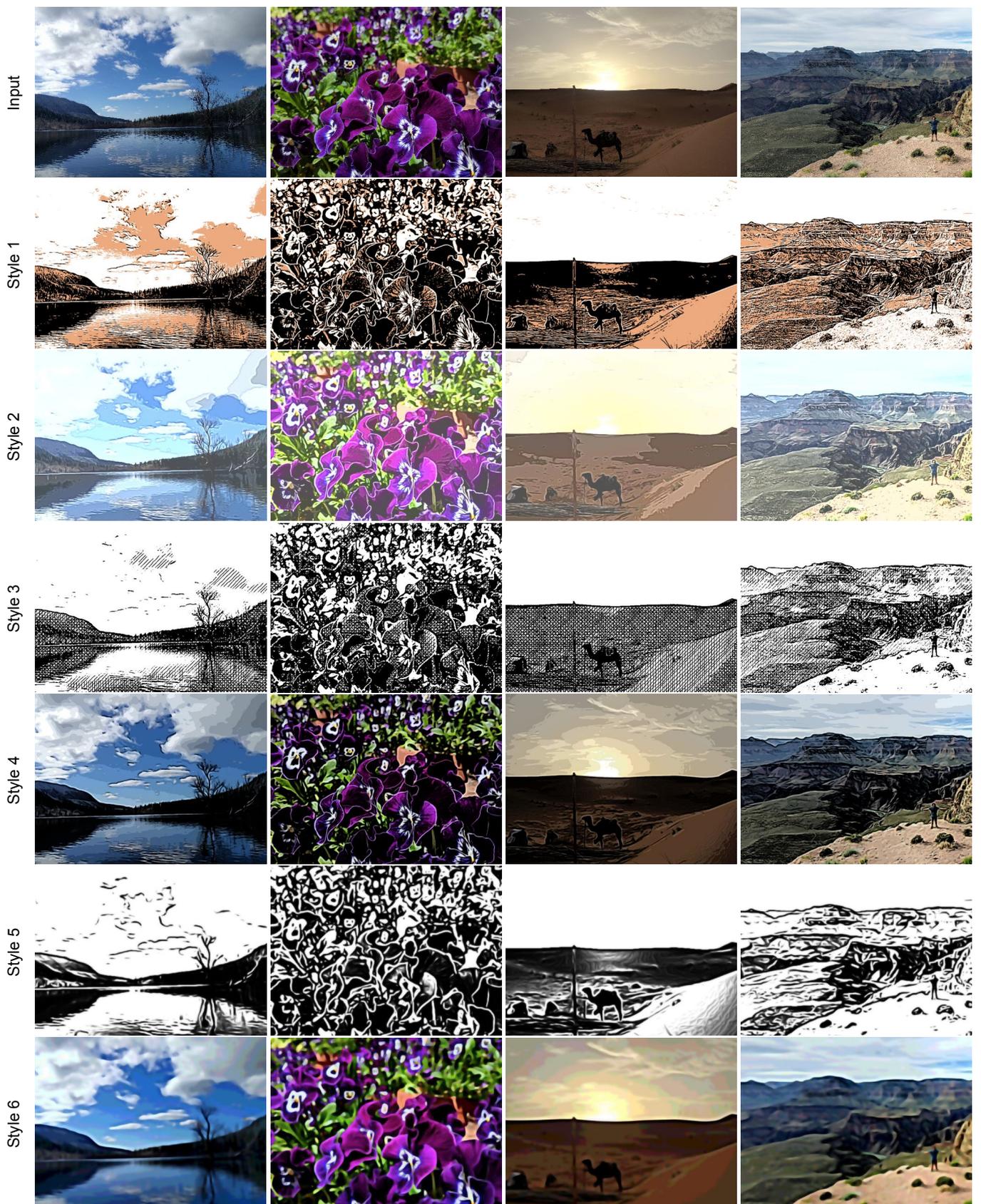


Fig. S8 Additional stylization examples: Input image on top and six different stylizations created with our tool. See details in Sec. 6.1. Readers are encouraged to zoom in aggressively (200% or more).