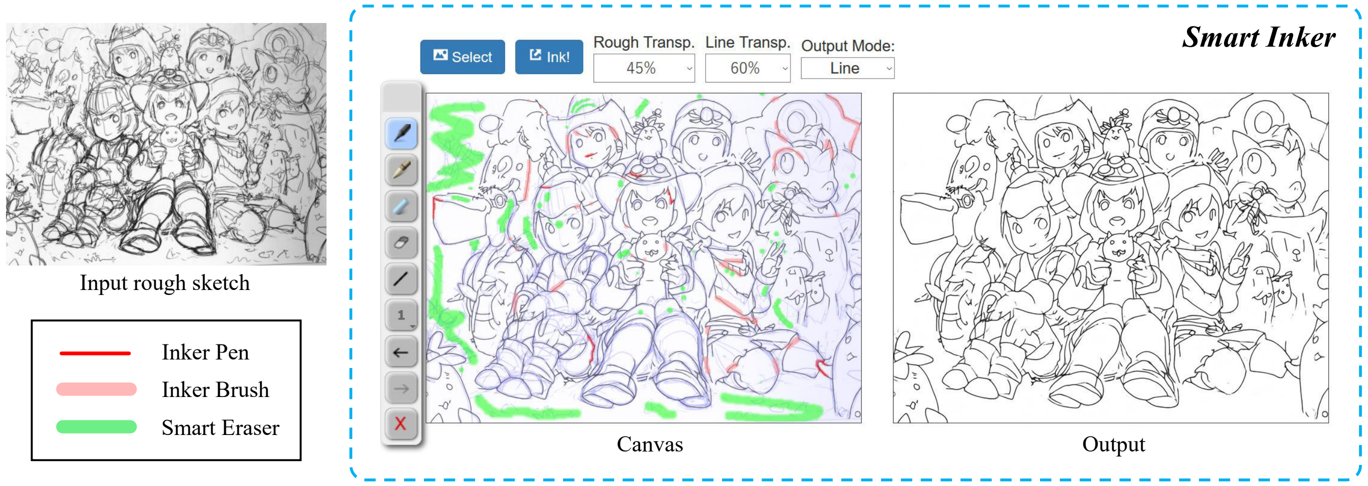
Real-Time Data-Driven Interactive Rough Sketch Inking

EDGAR SIMO-SERRA, Waseda University
SATOSHI IIZUKA, Waseda University
HIROSHI ISHIKAWA, Waseda University



Rough Transp. 45%
Line Transp. 60%
Output Mode: Line

Fig. 1. Example of interactive rough sketch inking. On the left we show an input rough sketch and on the right our user interface with the completed line drawing. The left of the user interface shows the canvas with user edits overlaid; green indicates *smart eraser* edits, light red shows *inker brush* edits, and dark red shows *inker pen* edits. On the right of the user interface we see the final result. This example was done using a mouse, and not any specialized illustration hardware. The image is copyrighted by Krenz Cushart and is part of the Krenz's Artwork Sketch Collection 2004-2013.

We present an interactive approach for *inking*, which is the process of turning a pencil rough sketch into a clean line drawing. The approach, which we call the **Smart Inker**, consists of several "smart" tools that intuitively react to user input, while guided by the input rough sketch, to efficiently and naturally connect lines, erase shading, and fine-tune the line drawing output. Our approach is data-driven: the tools are based on fully convolutional networks, which we train to exploit both the user edits and inaccurate rough sketch to produce accurate line drawings, allowing high-performance interactive editing in real-time on a variety of challenging rough sketch images. For the training of the tools, we developed two key techniques: one is the creation of training data by simulation of vague and quick user edits; the other is a line normalization based on learning from vector data. These techniques, in combination with our sketch-specific data augmentation, allow us to train the tools on heterogeneous data without actual user interaction. We validate our approach with an in-depth user study, comparing it with professional illustration software, and show that our approach is able to reduce inking time by a factor of 1.8×, while improving the results of amateur users.

CCS Concepts: • **Computing methodologies** → **Image manipulation**; • **Human-centered computing** → *Gestural input*; • **Applied computing** → *Fine arts*;

Authors' addresses: Edgar Simo-Serra, Waseda University, esimo@aoni.waseda.jp; Satoshi Iizuka, Waseda University, iizuka@aoni.waseda.jp; Hiroshi Ishikawa, Waseda University, hfs@waseda.jp.

© 2018 Copyright held by the owner/author(s). Publication rights licensed to the Association for Computing Machinery.
This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *ACM Transactions on Graphics*, https://doi.org/10.1145/3197517.3201370.

Additional Key Words and Phrases: inking, interaction, sketching, line drawing

ACM Reference Format:
Edgar Simo-Serra, Satoshi Iizuka, and Hiroshi Ishikawa. 2018. Real-Time Data-Driven Interactive Rough Sketch Inking. *ACM Trans. Graph.* 37, 4, Article 98 (August 2018), 14 pages. https://doi.org/10.1145/3197517.3201370

1 INTRODUCTION

"1. The inker's main purpose is to translate the penciller's graphite pencil lines into reproducible, black, ink lines.
2. The inker must honor the penciller's original intent while adjusting any obvious mistakes.
3. The inker determines the look of the finished art."
— Gary Martin, *The Art of Comic Book Inking* [1997]

Although the role of computers in illustration has grown significantly in the last decade, pencil and paper sketching still remains the dominant way most illustrators brainstorm or begin new projects. Once a rough sketch is completed, it is usually inked, either digitally or by pen, to obtain a line drawing which is amenable to coloring, toning, or direct usage. The digital inking process consists of redrawing all the lines, while at the same time fixing mistakes or tweaking details, to obtain a digital line drawing. The process is usually done by employing costly digital pen tablets, and requires concentration for drawing accurate lines. In this work, we present a digital inking process, which allows for semi-automatic real-time interactive creation of line drawings, while not requiring accurate

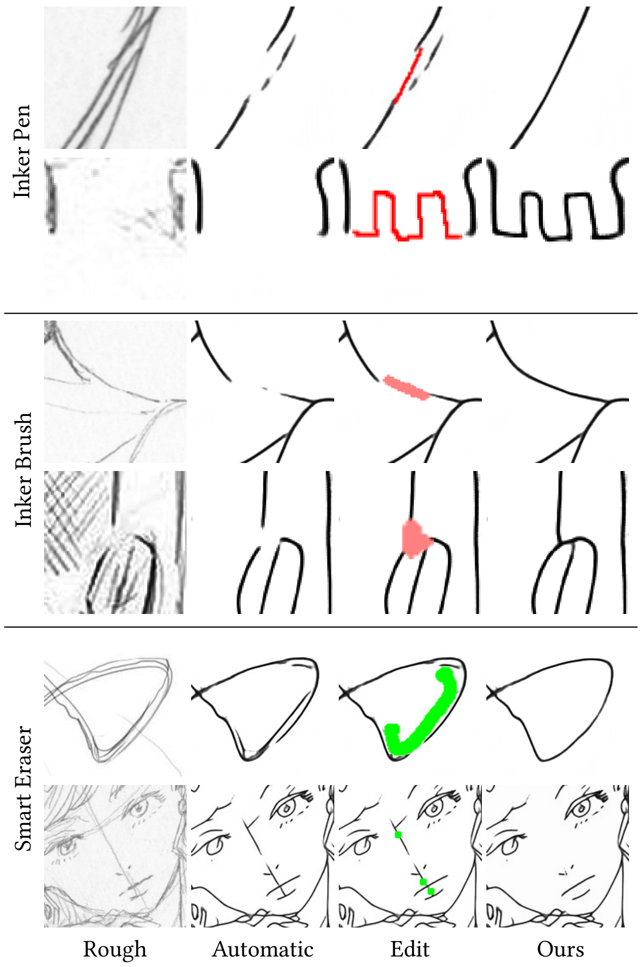


Fig. 2. Examples of our three proposed tools. The first two rows depict the *inker pen tool* (dark red), while the middle two rows depict the *inker brush* tool (light red), and the last two rows depict the *smart eraser tool* (green).

strokes, allowing amateurs to accurately ink complex pencil drawings in minimal time. An example is shown in Fig. 1.

Our approach consists of three "smart" tools that intuitively react to user input, while guided by the input rough sketch, to efficiently and naturally connect lines, erase shading, and fine-tune the line drawing output. The tools are: an *inker pen*, for precise correction or addition of lines; an *inker brush*, for quick addition of lines; and a *smart eraser*, which erases lines while considering their connectivity. We will show that this approach allows amateur artists and naïve users to quickly and accurately ink complex scenes, on average 1.8× faster than conventional approaches. An overview of the different tools is shown in Fig. 2, where it can be seen that inaccurate edits can be used for effective inking.

These tools are designed in a data-driven way, trained by training data and simulated user edits without explicitly hard-coding their behaviour. They are fully convolutional networks that take two kinds of inputs: the image of the input rough sketch and the user input. The ability to combine the two is similar to the recently proposed interactive neural network approaches [Sangkloy et al. 2017; Zhang et al. 2017] for image colorization, where the user can iteratively adjust the output by adding more user hints. When training with pairs of greyscale and color images, user hints are automatically generated by sampling from the ground truth image color. While these simple hints work well for interactive image colorization, where information is dense, it does not work for sparse inputs such as rough sketches and line drawings. Furthermore, inking requires more precise user edits than the hints used in image colorization. Because of these limitations, we propose a novel user edit simulation approach. An overview is shown in Fig. 3.

To train the networks, we require two novel techniques to create training data. One is the simulation of vague and quick user edits. The other is a line width normalization, which allows for training the model with a larger heterogeneous dataset in which the width of the lines is not consistent among samples.

For the latter, we developed a framework for learning a line normalization operator directly from data using neural networks. Unlike standard morphological operators which can be used for the line normalization task, our approach allows processing images without converting them to binary images as a pre-processing stage, and outputs smooth anti-aliased figures. We show that for complex line art this obtains much better performance than standard morphological operators. Our learned line normalization operator plays a fundamental role in the training of our network. The same framework can also be used as a post-processing step to tweak the final line width of the line drawings.

Our model is also inspired by recent developments in sketch simplification, in which using an encoder-decoder architecture fully convolutional network has allowed for fast and accurate sketch simplification of complex rough sketches [Simo-Serra et al. 2016]. We modify the model to allow for interactive editing, and simplify its design in order to reduce the computation time to roughly 37% of the original model.

A common issue creating line drawings with neural networks is blurry outputs. By using our line normalization framework, in combination with a simple weighted L_1 loss, we are able to obtain crisp results without having to resort to post-processing techniques [Simo-Serra et al. 2016] nor complex optimization frameworks [Simo-Serra et al. 2018]; this is rather critical for the interactive inking setting, as it allows for more natural and direct interaction.

We perform an in-depth evaluation of the different components of our approach, and provide results on a large number of challenging real world rough sketches drawn by professional artists. Additionally, we perform a user study in which we compare our approach with professional inking software, and show that, for amateur artists, our approach obtains better results an average of 1.8× faster than conventional approaches, and furthermore, users find our approach easier to use.

Our contributions are as follows: (1) An inking approach that allows for real-time interactive editing of high resolution rough sketches without the need of post-processing, with intuitive data-driven inking tools. (2) A framework for training such tools by simulating user edits and modifying the input rough sketch images. (3) A line width normalization technique for training with heterogeneous raster line sketches. (4) A study showing that amateur users are able to ink rough sketches 1.8× faster while improving the results.

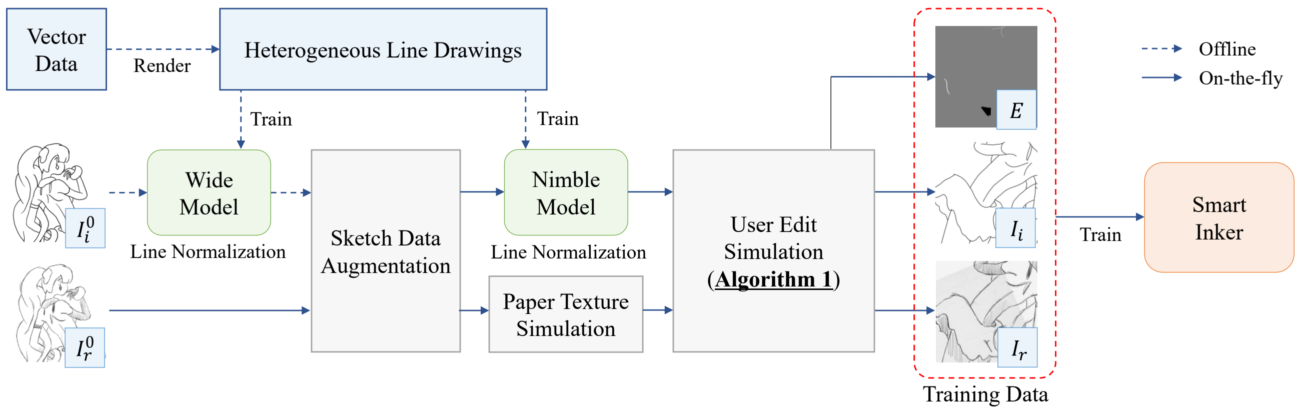


Fig. 3. An overview of our approach for training data-driven inking tools based on simulating user edits and modifying the input rough sketch images.

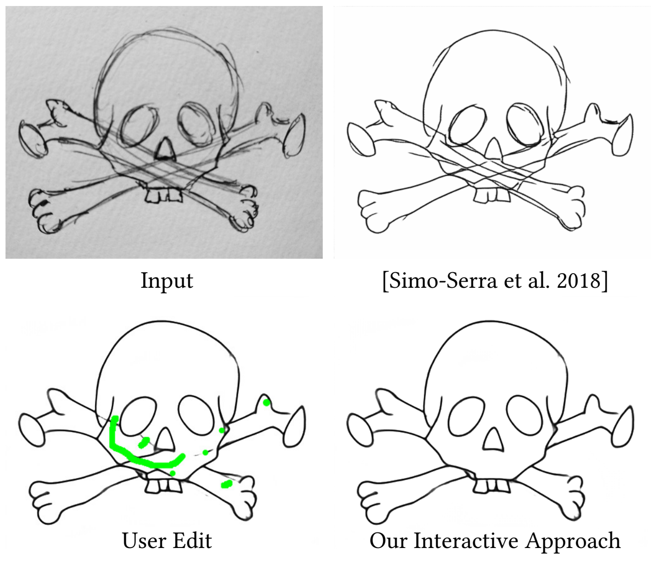


Fig. 4. Our approach can be used to iteratively improve results in a simple fashion, enabling it to handle foreground-background intersections which are common in rough sketch drawings. The *smart eraser* strokes drawn by the user are shown in green. In this skull and bones example, the bones that intersect the skull can be erased in a few seconds as it is not necessary to be very accurate with the smart eraser tool: it will automatically extend to fully erase intersecting lines.

2 BACKGROUND

Initial line drawing processing techniques have focused on the processing of vector images. With the availability of individual stroke data, it is possible to assist the user in creating line drawings by using geometric constraints [Igarashi et al. 1997], fitting Bézier curves [Bae et al. 2008], or heuristically merging strokes [Grimm and Joshi 2012]. However, all these approaches rely on having the data and order of the individual strokes, which is not possible when drawing with pencil drawn rough sketches, greatly limiting their applicability. Other works have relaxed the requirement that the drawing order is available, and have been applied to simplify vector images [Fišer et al. 2015; Lindlbauer et al. 2013; Liu et al. 2015; Orbay and Kara 2011; Shesh and Chen 2008]. Furthermore, approaches to directly convert raster images to vector images using optimization processes have been developed [Chen et al. 2013; Favreau et al. 2016; Hilaire and Tombre 2006; Noris et al. 2013], but generally have to rely on an initial pre-processing step to convert the input rough sketch to a binary image. The quality of this pre-processing step becomes a critical bottleneck for the performance of this algorithm. Additionally, the optimization step itself is very costly, and does not allow for real-time processing. Our approach is able to handle rough raster sketches directly without pre-processing and allows for editing in real-time.

Recently, an interactive vectorization approach for natural images related to our approach has been developed [Xie et al. 2017]. This approach exploits the rich gradients, color information, and contour maps of the natural images, and allows editing of the vector output by performing merge and split operations on closed contour regions based on detected edges. While exploiting the properties of natural images gives good results, the proposed approach no longer becomes applicable to rough sketch images. In contrast, our approach is data-driven and based on simulating user edits focusing on rough sketch images.

Approaches based on neural networks have been proposed for the sketch simplification task [Simo-Serra et al. 2018, 2016] on raster rough sketch images. These approaches focus on fully automatic processing, and on simple rough sketch input images with little or no pencil shading nor areas that require editing. In contrast, in this work, we tackle the inking task. The difference is that, in general, the input rough sketch has mistakes or areas that are *meant* to be corrected, and thus are not amenable for fully automatic approaches. Although we also base our approach on a neural network, we propose a novel approach for simulating user edits that allows our model to be used interactively to ink rough sketches in real-time. An example is shown in Fig. 4, in which the foreground and the background are drawn in an overlapping manner and makes fully automatic approaches fail. Although it is possible to edit the output directly with an illustration software, this requires accurate edits, while our approach works directly with rough and vague user inputs, automatically connecting lines and erasing connected areas.

In this work, we need *line normalization* for efficient learning and to avoid blurred output lines. Here, we would like to make the width of all the lines in the training set the same, without changing the underlying line structure, so that the learning process is not distracted by various thickness of the lines. We do this by learning morphological operators from data. Morphological operators have many usages in image processing, from character recognition [Ahmed and Ward 2002] to segmentation of retinal blood vessels [Mendonca and Campilho 2006]. Most algorithms employed are based on theoretical principals to determine how to simplify small image patches, and are applied iteratively until convergence [Jang and Chin 1990; Lam et al. 1992; Zhang and Suen 1984]. However, these theoretical principles do not necessarily correspond to human perception, and can lead to unnatural results. While skeletonization approaches can be applied for line normalization, they are based on look-up tables for 3×3 binary image patches and are also generally limited to outputting binary images, although there also exists research on directly handling greyscale images [Chatbri and Kameyama 2014; Dyer and Rosenfeld 1979]. We will show how our line normalization framework is able to conserve the underlying line structure, allowing for high performance, while producing anti-aliased outputs. These improvements will allow robust training of our inking network by normalizing the training line drawings.

Recently, neural networks have been used for interactive colorization of images [Sangkloy et al. 2017; Zhang et al. 2017]. In these works, a greyscale image and a user hint image are concatenated and inputted to a neural network which colorizes the greyscale image following the user hints. Afterwards, the user edits the hint image based on the output, and processes the input greyscale image again with the newly modified hint image. This alternation is done in an iterative fashion, and gives the user a certain degree of control over the resulting image colorization. The colorization model is trained by showing it parts of the ground truth image colors as user hints. We build our framework on this approach. However, unlike natural images, line drawings and rough sketches are too sparse to allow for such simple training approaches as showing regions of the ground truth as user hints. We propose a novel approach for simulating user edits for training data. To represent three categories of user edits, we design three tools, *inker pen*, *inker brush*, and *smart eraser*, that the users would have at their disposal for interactive inking, and simulate the use of these tools so that the neural networks can learn the relationship between the user interaction and intended edits. Unlike their simple counterparts in standard illustration software, these tools represent user intention, rather than a precise edit, and use the input rough sketch to interpret the user manipulation. Thus, they consider the rough sketch region and are able to connect and smooth lines or erase connected line segments in a coherent manner. We provide in-depth evaluation of our approach and show that, unlike simple user hint approaches, our approach allows for natural and high performance inking of challenging rough sketch images. The user study shows that 80% of the users find our approach easier to use than professional inking software, and they are able to ink the rough sketches roughly 1.8× faster.

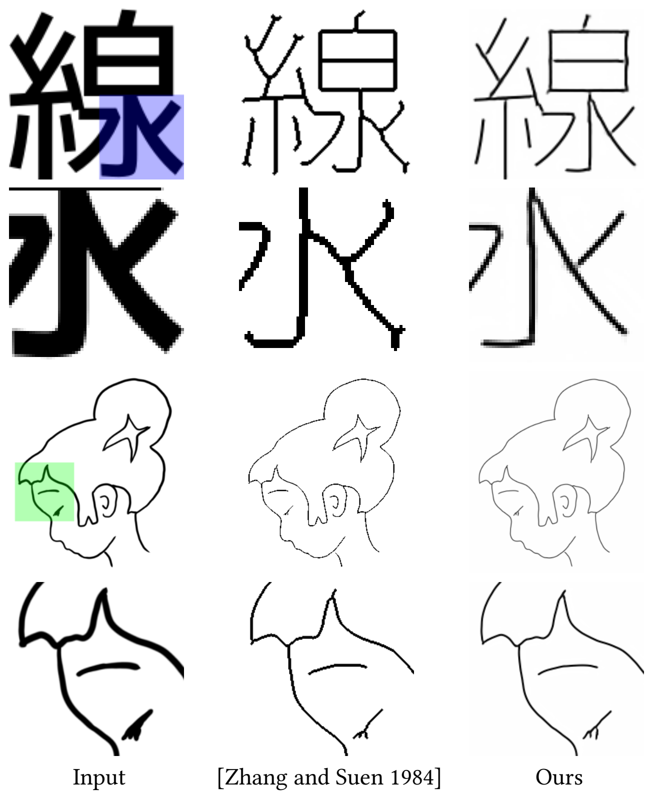


Fig. 5. Results of normalizing the line width of a raster line drawing in comparison with standard morphological operators [Zhang and Suen 1984]. We can see that our approach is able to accurately thin the lines while anti-aliasing without deforming or distorting the different strokes composing the line drawing.

3 LINE NORMALIZATION

In order to normalize heterogeneous line data for training Smart Inker, we propose a data-driven framework to normalize the stroke width of raster line drawings from vector data.

3.1 Framework

The standard approach to normalize the width of line drawings is based on morphological operators, which operate on binary images. These are based on look-up tables for 3×3 binary pixel patches [Zhang and Suen 1984], and it is not feasible to scale them up to larger and more complex patterns. Furthermore, binary image outputs are generally not desired for most applications. For this reason, we propose to learn the heuristics directly from large amounts of vector data, using very simple convolutional neural networks.

Our framework consists of using vector data and manipulating the line width to create input and output training pairs, which are amenable for training the convolutional neural networks. In particular, we focus on line normalization usage case, in which the line width of the target output image is fixed for all strokes. This can be implemented with traditional morphological operators as skeletonizing the lines to one pixel width, then dilating them to the desired thickness. Examples of our framework used for line

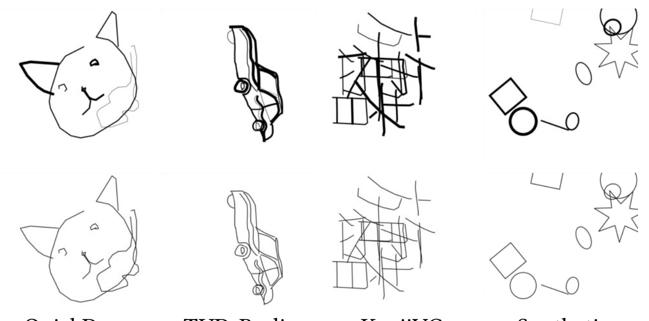


Fig. 6. Line width normalization network training examples. Top row shows inputs and bottom row shows ground truth images. These are used to train the line width normalization network in a supervised fashion.

normalization in contrast with existing approaches are shown in Fig. 5.

3.2 Models

We propose two fully convolutional network models: a wide model with 9 layers, and a nimble model with 5 layers. The wide model is designed for pre-processing large datasets offline, while the nimble model is meant to be used as part of the per-sketch data augmentation performed during training as shown in Fig. 3. For all layers except the last of each model, we use batch normalization [Ioffe and Szegedy 2015] with a Rectified Linear Unit transfer function [Nair and Hinton 2010]. For the last layer of each model, we use a Sigmoid transfer function to ensure that the output is in the $[0, 1]$ range.

We base our wide model on a fully convolutional network with 9 layers. The first layer uses a 9×9 pixel kernel, while the rest of the layers use 3×3 pixel kernels. The number of convolutional filters used is constant throughout the entire network and is set to 64. This allows the network to compute each output pixel from a 25×25 pixel region of the input.

The nimble model follows a similar approach, although it is reduced to 5 layers and only uses 32 convolutional filters throughout the network. As with the wide model, the first layer uses a 9×9 pixel kernel while the rest use 3×3 pixel kernels. The model computes each output pixel from a 17×17 pixel region of the input.

3.3 Training

We train these models using mixed sources of data. In particular, we use the TUD-Berlin dataset [Eitz et al. 2012], the QuickDraw dataset [Ha and Eck 2018], the KanjiVG dataset[1], and synthetic shapes.

For all datasets, we align all the vector drawings to the top-left corner and uniformly scale them to have a maximum size of 255. We resample all the strokes with 1 pixel spacing and then simplify them using the Ramer-Douglas-Peucker algorithm [Douglas and Peucker 1973; Ramer 1972] with an epsilon value of 2.0.

We sample from the various datasets with different frequency. In particular, the synthetic shapes are only 1/3 as likely to be used in comparison with the other datasets, which are used with equal probability. Instead of using a single drawing per training sample, we use a variable number depending on the level of detail of vector images in each dataset, which we heuristically determined by visual inspection of the average image complexity. They are also rotated and positioned randomly in training raster image. In particular, for the TUD-Berlin dataset, we only render one vector image between 200 and 500 pixels in resolution. For the QuickDraw dataset, we render up to two vector images between 200 and 500 pixels in resolution. For the KanjiVG dataset, we render up to three vector images between 50 and 300 pixels in resolution.

Training is done with the ADADELTA variant of stochastic gradient descent [Zeiler 2012], and all models are trained to normalize the line strokes to a constant width of 2 pixels. We train the wide model with inputs that have lines of variable thickness between 0.5 and 10 pixels, while the nimble model is trained with variable thickness between 0.5 and 3 pixels. The input patches are all generated to be 512×512 pixels, and we train with a batch size of 8. We use early stopping is used to avoid overfitting, and train for $20,000$ iterations. Example training data used for line normalization can be seen in Fig. 6.

3.4 Extensions

We note that this framework is trivially extensible to other morphological operators, such as dilation or erosion. In those cases, instead of generating fixed width line strokes, the width can be set relative to the input width, allowing training of models that, for example, increase or decrease the line width by a fixed amount.

4 INTERACTIVE ROUGH SKETCH INKING

We base our inking model, which we denote **Smart Inker**, on a convolutional neural network with extensive modifications in order to allow interactive editing and high real-time performance. In order to train the model for interactive editing, we propose a framework for simulating user edits in which we also modify the input rough sketch training images, nudging the model to make use of the simulated edits.

4.1 Interactive Inking

From an algorithmic point of view, inking can be seen as a mapping of the input rough sketch I_r to the output ink line drawing image I_i, where $I_r, I_i \in \mathbb{R}^{w \times h}$ are greyscale images with width w and height h. Instead of approximating this mapping directly, we consider the mapping of a pair (I_r, E) of rough sketch and user inputs to I_i.

In the greyscale images, pure white pixels have the value 1 and pure black pixels have value 0. We also represent the user input by a greyscale image $E \in \mathbb{R}^{w \times h}$, which is amenable to be used with convolutional operations, while being expressive enough to represent many different types of user edits.

We define three different basic editing tools: *inker pen*, *inker brush*, and *smart eraser*. The inker pen is used to ink lines, the inker brush is used to connect incomplete strokes, and the smart eraser is to erase unnecessary lines. Unlike standard tools, these will be trained with data in order to generate clean line drawings. We set each "pixel" of

[1]http://kanjivg.tagaini.net/

Table 1. Overview of the model architecture. We abbreviate Convolution with "Conv." and Sub-Pixel Convolution with "Sub-Conv.". We specify layer irregularities in the notes column. When the same layer is repeated several times consecutively, we indicate this with the number of times in parenthesis. Layers are grouped by internal resolution.

Layer Type	Output Resolution	Notes
Input	$2 \times W \times H$	Rough sketch and user edit
Conv.	$128 \times W/2 \times H/2$	9×9 kernel, refl. pad, stride 2
Conv. (×4)	$128 \times W/2 \times H/2$	
Conv.	$256 \times W/4 \times H/4$	Stride 2
Conv. (×3)	$256 \times W/4 \times H/4$	
Conv.	$512 \times W/8 \times H/8$	Stride 2
Conv. (×2)	$512 \times W/8 \times H/8$	
Conv. (×6)	$256 \times W/8 \times H/8$	
Dropout	$256 \times W/8 \times H/8$	20% spatial dropout
Sub-Conv.	$64 \times W/4 \times H/4$	
Conv.	$64 \times W/4 \times H/4$	
Sub-Conv.	$16 \times W/2 \times H/2$	
Conv.	$16 \times W/2 \times H/2$	
Sub-Conv.	$4 \times W \times H$	
Conv.	$1 \times W \times H$	No BN, sigmoid

E to be a particular tool or to perform no action:

$$E(u,v) = \begin{cases} 0 & \text{if no action} \\ \phi_p & \text{if \textit{inker pen} tool} \\ \phi_b & \text{if \textit{inker brush} tool} \\ \phi_e & \text{if \textit{smart eraser} tool} \end{cases}, \tag{1}$$

where ϕ_p, ϕ_b, and ϕ_e are real number hyper-parameters. We note that, unlike using standard illustration tools as a post-processing stage, we will train our inking model to learn to incorporate these tools in the learning process, allowing the model to adjust the output to match the user edits in a "smart" fashion.

We consider the usage scenario in which the user first performs inking on a rough sketch in fully automatic fashion before iteratively and interactively correcting the output of the model.

4.2 Smart Inker Model

Our model consists of a fully convolutional encoder-decoder network with 24 layers that decreases the resolution in three stages down to 1/8 of the original size. Afterwards, the output is restored to the original size in three stages with sub-pixel convolutions [Shi et al. 2016], which allow doubling the resolution at each stage. Following the standard practices in convolutional networks, after each convolutional layer we apply Batch Normalization (BN) [Ioffe and Szegedy 2015], followed by a Rectified Linear Unit (ReLU) transfer function [Nair and Hinton 2010]. The convolutional layers use 3×3 pixel kernels with 1×1 pixel 0-value padding to keep the internal resolution constant. As an exception, the first layer uses a 9×9 pixel kernel with 4×4 pixel reflection padding to avoid artefacts in the input caused by using 0-value padding. We also employ spatial dropout [Tompson et al. 2015] before the first sub-pixel convolution layer, which is an extension of dropout [Srivastava et al. 2014] in which instead of randomly 0-valuing pixels, entire feature maps are randomly set to 0. This avoids nearby pixel values being used to fill in 0-value pixels that is typical due to the heavy correlation of neighbour values. An overview of the model is shown in Table 1.

Table 2. Comparison of computation time on a machine with a Intel(R) Core(TM) i7-6800K CPU (3.40GHz) and GeForce GTX 1080 GPU with LtS [Simo-Serra et al. 2016]. The average timing of 100 trials is shown.

Approach	Parameters	1024^2px	1512^2px	2048^2px	2560^2px
LtS	44,551,425	238.8ms	562.4ms	984.7ms	1.59s
Ours	12,795,169	89.9ms	225.5ms	382.7ms	592.9ms

The closest work to this model is that of [Simo-Serra et al. 2016] which consists of a 23 layer convolutional network. While our proposed model is similar in architecture, we would like to point out several major differences. First and most importantly, we consider as input both a rough sketch and user edit maps, which can be seen as a two channel image instead of a single channel greyscale image. Instead of using up-convolutions, we use sub-pixel convolutions which allow for faster computation and higher accuracy. Finally, we would like to note that we have optimized the entire model architecture to significantly increase the computational efficiency of the model while making no compromises on accuracy. An overview of computational efficiency can be seen in Table 2 in which our model obtains roughly 3× faster performance on a GPU, which results critical in allowing real-time usage of our model.

4.3 Simulating User Edits

One of the most important aspects of training Smart Inker is simulating realistic user edits. Unlike traditional software tools, which use an imperative design approach in which each tool is explicitly defined, here we have to define the inputs and outputs of a task, and train the model to learn the input to output mapping. In particular, we simulate edits for the three different proposed tools in a way to encourage the model to respect them when inking rough sketches. Furthermore, when simulating, we do not only generate the edit map E, but also manipulate and edit the input rough sketch I_r. An overview of the entire algorithm for simulating edits can be seen in Algorithm 1 and examples of simulated edits can be seen in Fig. 7.

For a rough sketch and a line drawing pair (I_r, I_i), we randomly simulate the different type of tools to generate a user edit map E. In particular, we simulate four different actions: inker pen tool, inker brush tool, simple eraser, and advanced eraser. Both the simple eraser and the advanced eraser simulated actions correspond to our smart eraser tool. In order not to be fully dependent on the edits, with 10% probability we use a fully zero (no-action) edit map E. Otherwise, we create a number of additive edits (inker pen tool and inker brush tool) following a binomial distribution $\binom{N_L}{\sigma_L}$ and a number of subtractive edits (smart eraser tool) following a binomial distribution $\binom{N_E}{\sigma_E}$. For each additive edit there is a 50% possibility of it being a simulated inker pen tool edit and 50% possibility of it being a inker brush tool edit. In the case of the smart eraser tool, each edit has 50% chance of being a simple eraser edit and 50% chance of

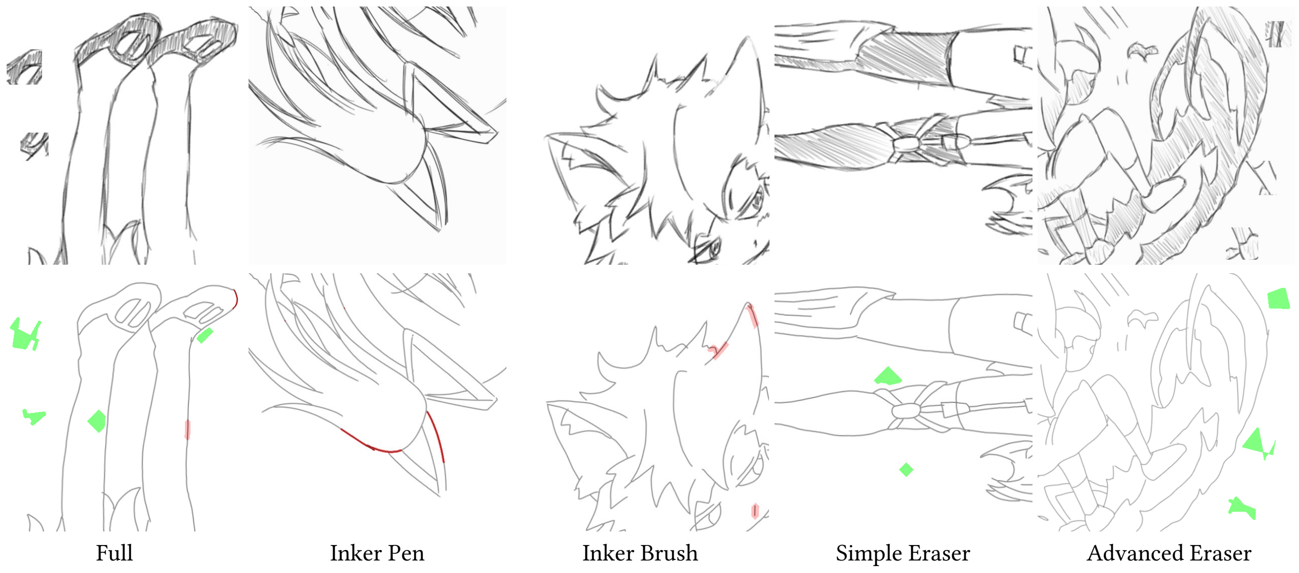


Fig. 7. Training examples with simulated user edits. In the top row we show the rough sketch input, and in the bottom row we show a linear combination of the simulated user edit and the input ground truth. We can see how, for the inker brush and advanced eraser simulation, not only is the user edit being simulated, but the input rough sketch is also being modified. This forces the model to learn to exploit the user edits and not rely solely on the input when inking. In particular, when simulating the inker brush tool we can soften or erase parts of the input, while when simulating the advanced eraser will copy parts of the rough sketch to simulate superfluous lines.

being an advanced eraser edit. In general, as the user edit simulation must be done a large amount of times during the training procedure, we prioritize keeping the simulation as simple and fast as possible.

4.3.1 Inker Pen Simulation: The inker pen tool is one of the easiest to simulate. Initially the ground truth line drawing is binarized using a threshold τ_L and a random point is sampled on a "line" pixel as an initial seed. From this seed, we perform a directional dilation operation using the inverse of the binary line drawing as a mask and add the result times ϕ_p to the user edit map. The directional dilation operation is implemented as a recursive function which accumulates the image gradient vector, and only propagates the dilation to pixels with relatively similar image gradients directions. In particular, we dilate randomly between 50 and 90 pixels. An overview of this algorithm can be seen in Algorithm 2. After the dilation, the lines are randomly distorted by either one-pixel dilation or erosion operation 30% of the time, and used to update the user edit map. This decreases the sensitivity to line width.

4.3.2 Inker Brush Simulation: While the inker pen tool is useful for accurately inking lines, it is overly dependent on the accurate strokes and not very suitable for amateur users. In order to also appeal to non-professionals, we propose a inker brush tool, which also allows completing lines, but unlike the inker pen tool, it completes lines in a fuzzy manner. Instead of relying on accurate user strokes, the inker brush leverages sloppy inaccurate user strokes to correct or add new lines based on the rough sketch input. We train the model to use this tool by first skeletonizing the ground truth line drawing thresholded by τ_L. We then choose a random point on the line drawing skeleton, and dilate this point randomly between 10 to 30 pixels using a version of the line drawing that has been randomly dilated between 3 and 7 pixels as a mask. This allows us to obtain a large fuzzy line centered on a single point. We then randomly translate the fuzzy line randomly by 0 to 2 pixels both horizontally and vertically, and add the result times ϕ_B to the user edit map. We found that just generating inaccurate fuzzy lines led to the network relying solely on the input rough sketch and ignoring the fuzzy lines due to their inaccuracy. For this reason, we also randomly erode the dilated non-translated line stroke by 0 to 5 pixels, apply a Gaussian filter with a random standard deviation between 0 and 1, and randomly interpolate the input rough sketch with this new mask. This causes part of the input rough sketch to whiten out and no longer allows the network to rely on the input, which has been partially erased, forcing the network to reconstruct the output based on the simulated inker brush tool input and nearby lines.

4.3.3 Simple Eraser Simulation: Simple simulation of the eraser tool is analogous to the inker pen tool, but for white space. First the ground truth line drawing is binarized using a threshold τ_E, and a random white coordinate is sampled. Afterwards, this coordinate is dilated between 10 and 50 pixels using the white space as a mask to create an initial eraser candidate, which is then randomly eroded between 0 and 20 pixels such that the boundary doesn't exactly match the mask. The result times ϕ_e is added to the user edit map.

4.3.4 Advanced Eraser Simulation: One issue with generating simple eraser user edits is that, when the model performs very well, it will have a tendency to ignore the eraser edits, and only focus on exploiting the rough sketch and additive edits. In order to force the model to heed the eraser edits, we randomly select a square patch between 20 to 40 pixels from the input rough sketch and

Algorithm 1 Overview of the training data generation procedure using simulated user edits.

```
function SIMULATE_USER_EDIT
    Input: $\emptyset$
    Ouput: rough sketch $I_r$, ground truth $I_i$, simulated edit $E$
    $I_r, I_i \leftarrow$ sample_training_data()
    $I_r, I_i \leftarrow$ data_augmentation($I_r, I_i$)
    $E \leftarrow$ zero_matrix()
    if 10% chance then                                    ▷ No user edit
        return $I_r, I_i, E$
    end if
    for $j = 1$ to $N_L$ do
        if $\sigma_L$ chance then
            $I_i' \leftarrow$ threshold($I_i, \tau_L$)
            if 50% chance then                    ▷ Simulate Inker Pen Tool
                $u, v \leftarrow$ random_line_coordinates($I_i'$)
                $E' \leftarrow$ dir_masked_local_dilation(($u, v$), $1 - I_i'$)
                $E' \leftarrow$ random_dilation_erosion($E'$)
                $E \leftarrow E + \phi_p E'$
            else                                  ▷ Simulate Inker Brush Tool
                $I_i' \leftarrow$ skeletonize($I_i'$)
                $u, v \leftarrow$ random_line_coordinates($I_i'$)
                $I_i' \leftarrow$ dilation($I_i'$)
                $E' \leftarrow$ masked_local_dilation(($u, v$), $1 - I_i'$)
                $E \leftarrow E + \phi_b$random_translation($E'$)
                $E' \leftarrow$ erosion($E'$)
                $E' \leftarrow$ gaussian_blur($E'$)
                $I_r \leftarrow I_r \odot E_s' - E_s' + 1$
            end if
        end if
    end for
    for $j = 1$ to $N_E$ do
        $I_i' \leftarrow$ threshold($I_i, \tau_E$)
        if $\sigma_E$ chance then
            if 50% chance then                    ▷ Simulate Simple Eraser
                $u, v \leftarrow$ random_whitespace_coordinates($I_i'$)
                $E' \leftarrow$ masked_local_dilation(($u, v$), $I_i'$)
                $E' \leftarrow$ erosion($E'$)
                $E \leftarrow E + \phi_e E'$
            else                                  ▷ Simulate Advanced Eraser
                $I_i' \leftarrow$ random_image_patch($I_i$)
                $I_i' \leftarrow$ erosion($I_i'$)
                $u, v \leftarrow$ random_whitespace_coordinates($I_i'$)
                $I_r \leftarrow$ min($I_r$, translate_to(($u, v$), $I_r'$))
                $E \leftarrow E + \phi_e$random_polyline(($u, v$))
            end if
        end if
    end for
    return $I_r, I_i$, clamp($E, -1, 1$)
end function
```

copy it to a random white space coordinate from a version of the ground truth line drawing eroded randomly by 0 to 20 pixels, which avoids significant overlap with the input rough sketch. At the same location, we render a vector polyline with a random thickness value between 3 and 10 pixels, which we use to update the user edit map such that the strokes overlap the copied input rough sketch patch. This forces the model to learn to properly erase marked areas of the image.

Algorithm 2 Overview of the recursive function used to locally dilate an image centered on a point (u, v) trying to conserve a constant image gradient while restricting the dilation to an image mask.

```
function DIR_MASKED_LOCAL_DILATION
    Input: image gradient $(\partial_u I, \partial_v I)$, mask $M$, image coordinates $(u, v)$, dilation amount $d$, dilated image $D$, accumulated gradient $(A_u, A_v)$
    Ouput: dilated image $D$
    if $d < 0$ then
        return $D$
    end if
    $\theta_A \leftarrow$ get_angle($A_u, A_v$)
    $A_u \leftarrow A_u + \partial_u I^{u,v}$
    $A_v \leftarrow A_v + \partial_v I^{u,v}$
    $\theta \leftarrow$ get_angle($A_u, A_v$)
    if angle_difference($\theta, \theta_A$) $\geq \tau_A$ then
        return $D$
    end if
    for all $(i, j) \in$ adjacent_pixels($u, v$) do
        $D \leftarrow$ dir_masked_local_dilation(($\partial_u I, \partial_v I$), $M$, ($u, v$), $d - 1$, $D$, ($A_u, A_v$))
    end for
    return $D$
end function
```

4.4 Interface

We build our interface as a combination of a GPU-based back-end with a simple web front-end as shown in Fig. 1-right. The user is presented with a both a large input area and output area. Following traditional inking practices [Martin et al. 1997], we display the rough sketch in blue with the output overlaid on top on the left. The user is then able to edit directly this image and the result is propagated to the output which is also displayed on the right. By having the front-end communicate asynchronously with the back-end, in addition to presenting a model that can process 1512 × 1512 images in 0.2s, we ensure the user is able to interactively edit large rough sketches in real-time.

5 TRAINING

We train our model on pairs of rough sketches and line drawings, in addition to using simulated user edits. We use line normalization networks for both offline and on-the-fly data normalization, which in combination with a weighted L_1 loss, allows obtaining crisp line drawing outputs, thus avoiding post-processing and allowing for more intuitive interactive editing by the user.

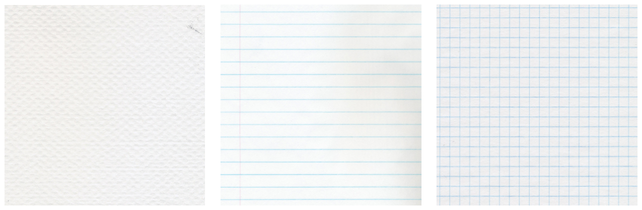


Fig. 8. Example of paper textures used to augment the training dataset by multiplying the per-pixel values with the input rough sketches.

5.1 Dataset

We have collected a new dataset of 288 rough sketches and line drawing pairs following the "inverse dataset construction" approach [Simo-Serra et al. 2016] drawn by six different illustrators. Unlike previous works, we collect much more complicated rough sketches that include large amounts of pencil shading and imperfections. These images are significantly more challenging than previous approaches, which is necessary when learning with simulated user edits. Some example patches extracted from the training dataset can be seen in Fig. 7.

5.2 Offline Data Normalization

One issue we found when creating a much larger and challenging dataset was that the thickness of the line drawings varies significantly between illustrator, and even between drawings, and leads to blurring in the outputs as we will show. For this purpose, we employ the line normalization model and normalize the dataset before initializing the training procedure. In particular, we use the wide model from §3.2, trained to normalize lines to 2 pixel width with anti-aliasing. First, we resize all the training images such that the total number of pixels is no more than 1512^2, then we process them with the line normalization model to obtain a more consistent and coherent dataset which minimizes the blurring in the output.

5.3 Per-Sketch Data Augmentation

Given the limited number of training samples, we perform large amounts of on-the-fly sketch-specific data augmentation to improve the generalization of our model. Following standard practices, we randomly rotate both the input and output training image pairs, in addition to random horizontal flipping and scaling. In particular, for each image, the amount of scaling is determined by the triangular probability distribution:

$$P(x; s_0) = \begin{cases} \frac{2(x-s_0)}{(1-s_0)^2} & \text{for } s_0 \leq x \leq 1, \\ 0 & \text{otherwise.} \end{cases} \quad , \tag{2}$$

where s_0 is the training patch size divided by the shortest edge of the image, and $x = 1$ would represent the original image resolution.

We also randomly modify the contrast of the input rough sketch between 0.8 and 1.2 with 1 being the original contrast. Each rough sketch input has a 15% chance of being modified with random noise chosen from either a normal distribution $\mathcal{N}(-0.05, 0.15)$ or uniform distribution $\mathcal{U}(-0.1, -0.2)$ that is filtered with a Gaussian kernel with $\sigma = 0.5$ to improve the robustness to input images.

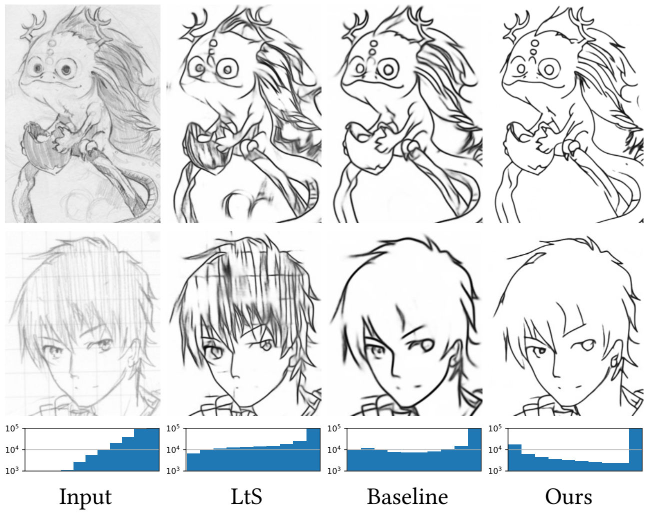


Fig. 9. Comparison of different models trained with and without the line normalization pre-processing and our weighted loss. We compare our full approach with LtS [Simo-Serra et al. 2016] without post-processing and a baseline that is exactly the same as our approach, except that it uses the MSE loss and no line normalization network. In the bottom row we show the histograms of the greyscale pixel values. Both the LtS and Baseline model show heavy blur in comparison to our approach. The top image is copyrighted by David Revoy (www.davidrevoy.com) and licensed under CC-by 4.0.

In order to further increase robustness to the types of papers used in rough sketching, we augment the rough sketches by combining them with images of different types of paper. In particular, we use 62 different images of different types of paper, and for each training sample, with 15% probability, randomly perform the following transformation:

$$I_r \odot \left(\frac{I_p}{b} + \frac{b-1}{b} \right) \quad , \tag{3}$$

where I_p is the paper texture, and $b \sim \mathcal{U}(1, 2)$. We note that, just like the rough sketches, we augment the paper textures I_p with random rotation, flipping, contrast adjustments and scaling. Figure 8 shows some examples of the different paper textures we use during training.

Finally, as we perform significant scaling as part of our on-the-fly data augmentation approach, we also use a small line normalization network (nimble model from §3.2) to renormalize the scaled line drawings back to 2 pixel line width after scaling. We use this smaller network as a fixed on-the-fly preprocessing stage during training, which improves the output of our model by further reducing blurring, as there is no ambiguity in the output line width, *i.e.*, all lines are of 2 pixel width.

5.4 Objective Function

We train the Smart Inker model S by minimizing

$$\theta^* = \arg\min_{\theta} \mathbb{E}_{I_r, I_i, E \sim \mathcal{D}} \left[L\left(S(I_r, E; \theta), I_i \right) \right] \quad , \tag{4}$$

where $\mathcal{D}$ is our simulate user edit function (Algorithm 1), and θ are the model parameters.

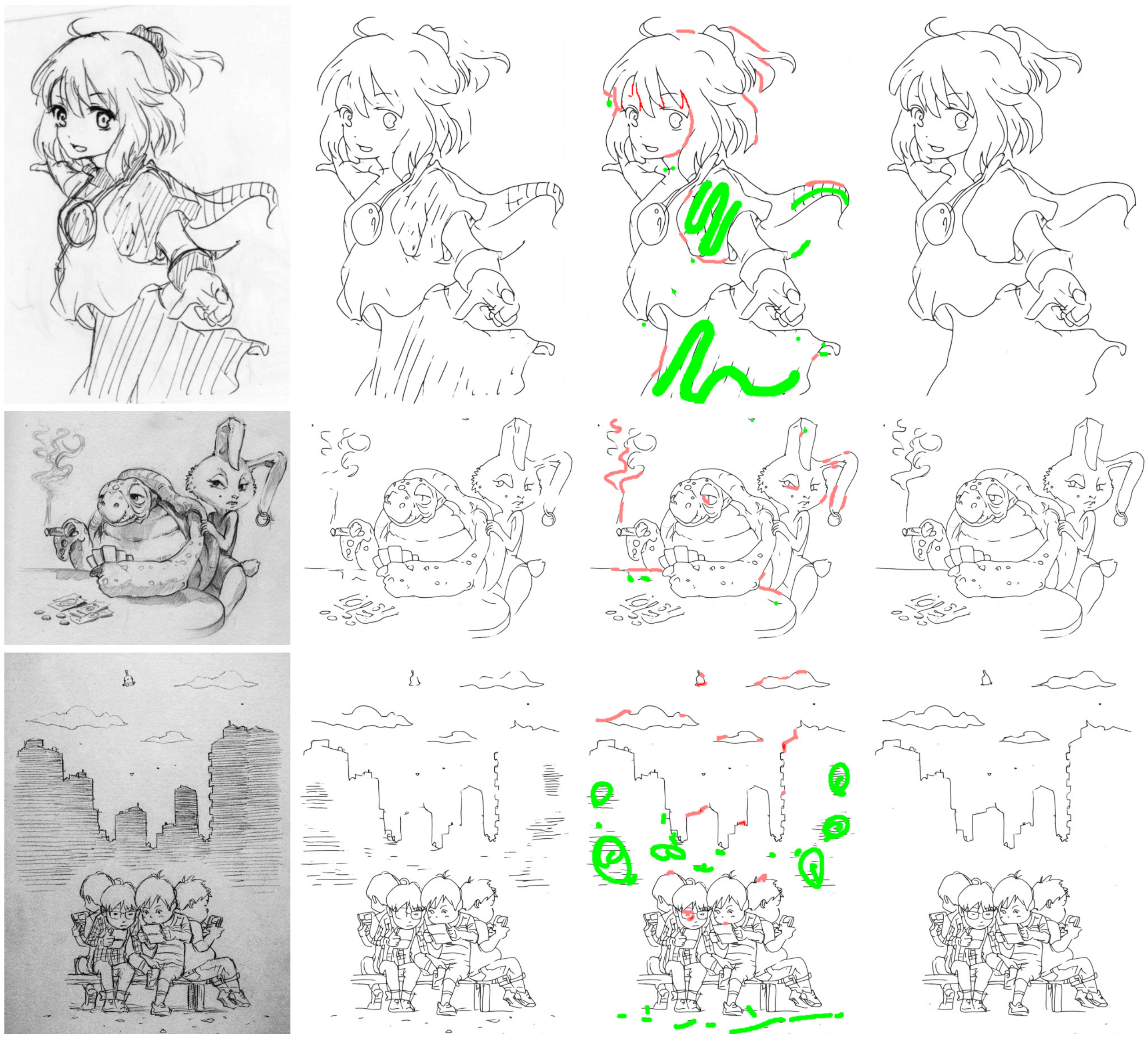


Fig. 10. Qualitative examples of our interactive inking approach. The second image from the top is copyrighted by David Revoy (www.davidrevoy.com) and licensed under CC-by 4.0, and the bottom image is copyrighted by Eisaku Kubonouchi.

As an objective function, we employ the weighted L_1 loss:

$$L(y, y^*) = \left|(y - y^*) \odot \left(1 + \gamma \left(1 - y^*\right)\right)\right| \quad , \tag{5}$$

where γ is a weighting hyper-parameter that controls how much importance is given to lines over white space.

We find that our weighted L_1 loss, in combination with the line normalization, allows us have clean anti-aliased outputs. This allows us to avoid additional post-processing of the output, which is important to giving more intuitive direct control to the user when performing interactive editing. A comparison with the standard training approach is shown in Fig. 9. Optimization is done using the ADADELTA [Zeiler 2012] variant of stochastic gradient descent, which does not need an explicit learning rate parameter.

6 EXPERIMENTS

We train our model with a batchsize of 27 with 512×512 pixel patches. We simulate a number of user edits taken from the binomial distribution $\binom{N_L}{\sigma_L} = \binom{N_E}{\sigma_E} = \binom{4}{0.4}$, which generates an average of 1.6 additive and 1.6 subtractive edits per image. When simulating the edits, we use thresholds of $\tau_L = 0.5$ and $\tau_E = 0.95$. The user edit map pixel values are set to be in the $[-1, 1]$ range, and, in particular, $\phi_p = 1$, $\phi_b = 0.5$, and $\phi_e = -1$. We train using Eq. (5) with $\gamma = 2$.

6.1 Interactive Editing

We show qualitative examples of user editing in Fig. 10. We note that as our interactive framework encourages fuzzy and inaccurate edits, the tools are set by default to provide large generous strokes. We can see that even very challenging rough sketches can be converted to clean line drawings.

6.2 Analysis of Data-Driven Tools

We perform an analysis of different usage cases of our proposed tools and show example results in Fig. 2. We can see how using the inker pen tool it is possible to correct ambiguous strokes or add novel complex lines to the drawing. The inker brush tool allows for simple and fast editing without using accurate strokes, and is able to correct mistakes in the original rough sketches. The smart eraser takes into account the context and allows removing the ambiguity of multiple rough sketch lines while conserving important strokes.

6.3 User Study

We quantitatively evaluate our approach with a user study, comparing with a Professional Tools (PT) for inking. In particular, we compare against Clip Studio Paint Ex, a professional illustration software package containing many advanced features such as jitter reduction, and optimized for pen tablet usage. We use a total of 10 diverse rough sketches and ask users to ink them into reproducible, black lines. Furthermore, we ask them to honor the original intent of the rough sketch while adjusting obvious mistakes. We point out it is not necessary to reproduce the pencil shading. The inking is done using a Wacom MobileStudio Pro 16, which is a professional mobile pen computer designed for illustration. We measure the time it takes for the participants to ink each rough sketch and have them answer a survey at the end. Each participant inks each rough image only once, inking half of the images with each tool. The order and what rough image they ink is randomized per user such that all images are inked the same number of times by each tool. A total of 10 people participated in the user study and were remunerated for their participation. Of the participants, 7 were female and 3 were male. Only 3 participants had significant experience in drawing, 3 had some drawing experience, and 4 were amateurs. The participants took on average 2.8 hours to complete the user study.

We compare the user's interaction time for both our approach and Professional Tools (PT) and show the results in Fig. 11-left. We compare the median values with the Mann-Whitney U test [Mann and Whitney 1947], a non-parametric statistical test, and obtain a p-value of 2.589×10^{-7}, indicating that our approach is indeed 1.8× faster than the Professional Tools (PT) on average. We show several examples from the user study in Fig. 12. In the first row we show the most complex image in the user study, in which users are able to ink the rough sketch 7.2× faster with our approach. In the second row we see a much more challenging rough sketch in which it is not clear what lines should be erased or kept. In this case, the interaction times are very similar, although we can see that our approach is able to output higher quality inking results in comparison with the amateur user.

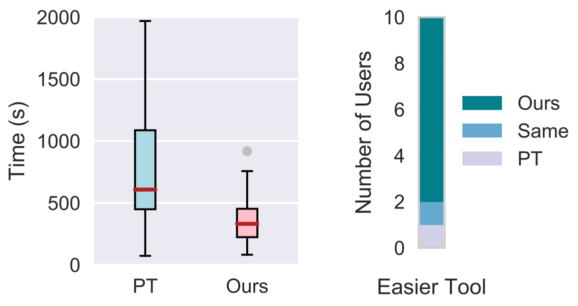


Fig. 11. Some results from our user study. On the left we show the interaction time for both the Professional Tools (PT) and our approach, and on the right we show the tool the users found easiest. The median value is highlighted in red, and outliers are shown in dark gray.

In our survey, we find that 8 of the 10 users find our approach easiest to use, while one user finds both approaches to be the same, and one prefers the Professional Tools (PT) as shown in Fig. 11-right. Half of the users find that our approach was easy to use, while the other half was neutral, and half believe their results are good with our approach while the other half considers their results to be fair. Of the three tools we propose, the users find the *inker brush tool* to be the easiest to use and the *inker pen tool* to be the hardest to use, although even then, 4 users find the inker pen tool easy to use and 6 consider it fair to use. All users believe that with practice they would be able to master Smart Inker and that it would be beneficial to add to professional illustration software.

For a complete report of the user study, please refer to the supplemental material.

6.4 Effectiveness of the User Edit Simulation

We compare with hint approaches similar to those used in recent colorization approaches [Sangkloy et al. 2017; Zhang et al. 2017], in which instead of our user simulated edits, we train the model by showing patches of the ground truth as hints. For each ground truth patch, we provide randomly either the line or the white space as a hint. In particular, we show it patches between 10 to 30 pixels in width and height. The rest of the approach remains the same as ours. We note that this approach is only able to learn the *inker pen tool* and *smart eraser tool*, as the *inker brush tool* is not represented in the ground truth line drawing. We show examples in Fig. 13. We find that the line tool does not automatically adjust the line width, and that the model is unable to learn to user the eraser tool. In fact, it learns that quite often the eraser tool hints are delineated by lines, which leads the model to often draw a line "halo" around the eraser edits. Our approach does not have this issue.

6.5 Comparison with Existing Automatic Approaches

We compare against recent approaches on challenging scanned rough sketches in a fully automatic approach *without using interactive editing*. In particular, we compare against [Favreau et al. 2016], LtS [Simo-Serra et al. 2016], and [Simo-Serra et al. 2018]. Results are

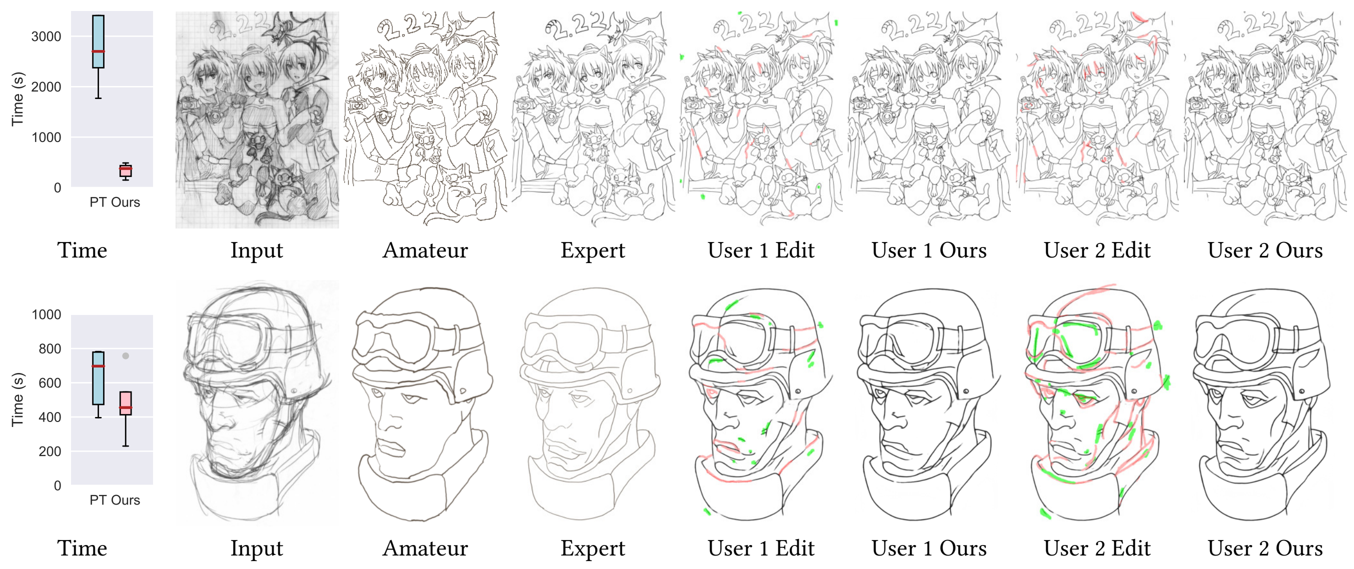


Fig. 12. Qualitative examples of images inked by users in our user study. We show the time taken by the different users, the input rough sketch, two inking done with Professional Tools (PT) with an example from an amateur and expert user, and finally two different examples of inking with our approach. The bottom image is copyrighted by Krenz Cushart and is part of the Krenz's Artwork Sketch Collection 2004-2013.

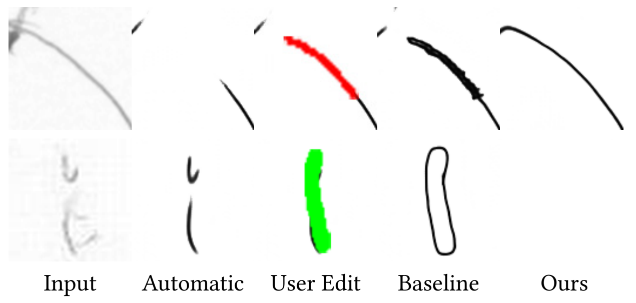


Fig. 13. Comparison with a baseline which is trained using parts of the ground truth line sketches as hints. Top row shows an example with the *inker pen tool*, while bottom row shows an example of the *smart eraser* tool.

shown in Fig. 14. For the approach of [Favreau et al. 2016], we pre-processed the images in order to obtain the results shown. We find that the approach of LtS, even with post-processing, tends to produce outputs that are fairly hard to make out for complicated rough sketches. The approach of [Simo-Serra et al. 2018] appears to perform better, but still performs slightly worse than our approach. Of the four approaches, only ours allows for interactive editing which we find important to obtain higher quality results.

6.6 Vectorization

It is common to use vector instead of raster representations when handling line drawings. We compare the results of using standard vectorization software directly on the rough sketch input and the inking results of our approach in Fig. 15. The vectorization of the rough sketch results in a noisy and dirty vector output, while the vectorized output of our approach remains a clean line drawing.

6.7 Changing the Line Width

We also show an illustrative example of how our line normalization framework can be used to change the line width in Fig. 16. In particular, we thicken the line by two pixels using a wide model (§3.2) and compare with standard binary dilation. We can see that our approach is able to accurately thicken the line while conserving the underlying structure and anti-aliasing the output.

7 LIMITATIONS AND DISCUSSION

We have proposed a semi-automatic real-time interactive framework for the inking of rough sketches. While our approach allows for faster inking, up to 7.2× improvement on complicated images, and improvements for amateur users, we found that for simple images with few lines, such as shown in Fig. 17, there is no statistically significant difference in interaction time. Our model is best suited for complicated images that take a long time to ink by hand.

One important aspect of our approach is that each output pixel is computed using a 241×241 pixel region of the input image and user edit. Although the input pixels spatially located near the output pixels are given more importance due to the nature of concatenating convolution operators, there are still cases in which the user edits affect regions of the output line drawing that the user did not intend to manipulate. In Fig. 18 we can see a case in which the user intends to connect the bottom line with the inker brush. Although they are successful at connecting the bottom line, the line above is partially erased. While this can be solved with additional edits, it can lead to mistakes and frustrate users. Additional training data might be a way to minimize this issue.

Finally, while our model is able to provide good general inking results, providing improvements for amateur users, and permits

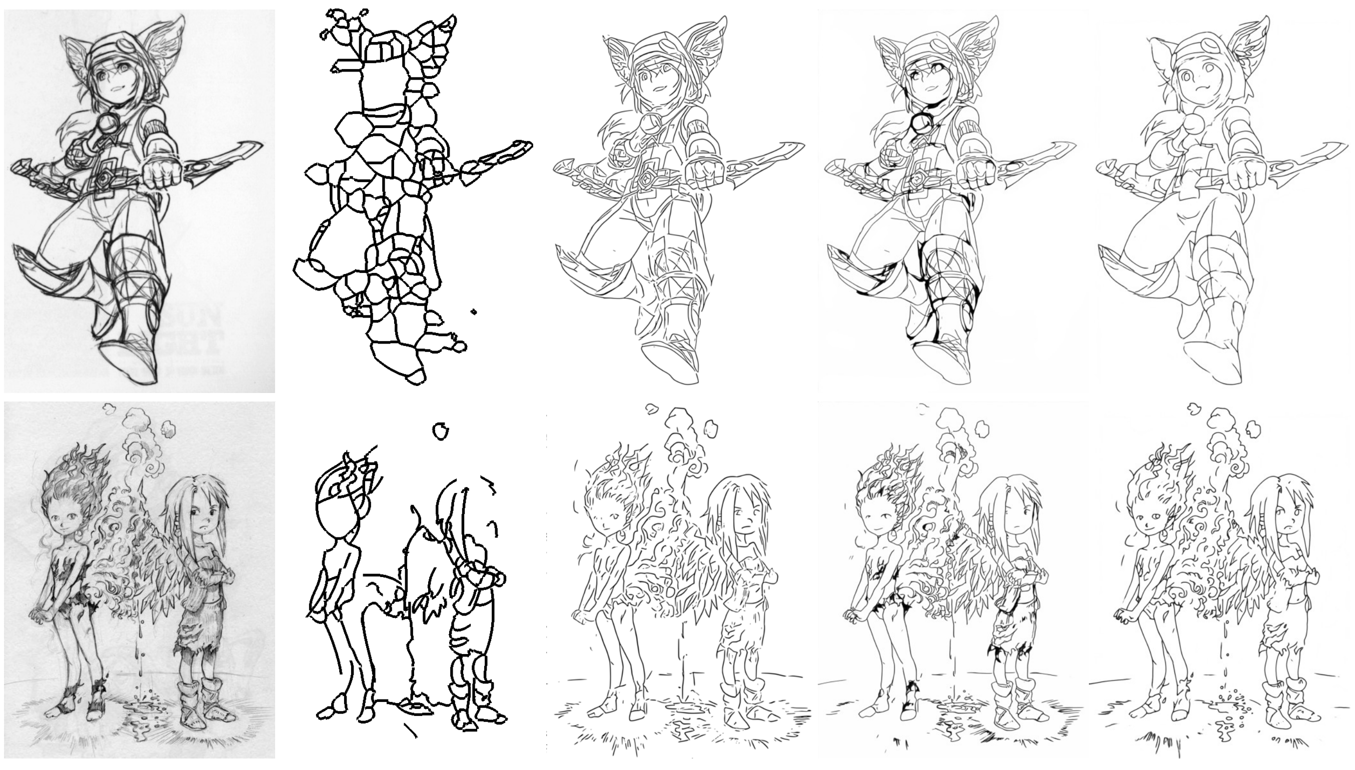


Fig. 14. Comparison against recent fully automatic sketch simplification approaches. We note that for the approach of [Favreau et al. 2016] we performed significant hyper-parameter tuning and used tone curves to clean up the rough sketch input in order to obtain the shown outputs. Of all the approaches, only ours allows for interactive editing. The top image is copyrighted by Krenz Cushart and is part of the Krenz's Artwork Sketch Collection 2004-2013, and the bottom image is copyrighted by David Revoy (www.davidrevoy.com) and licensed under CC-by 4.0.

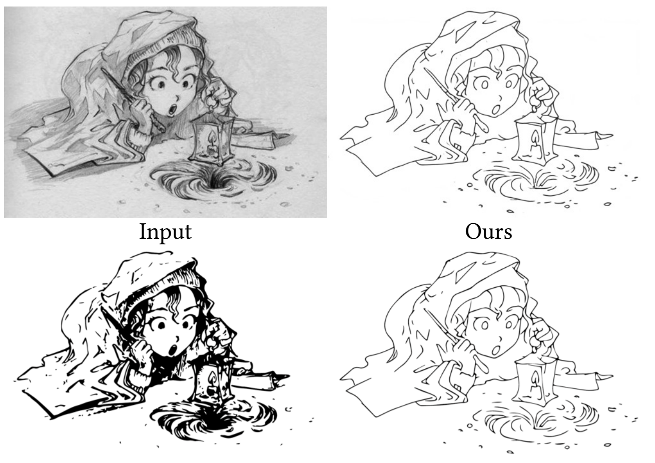


Fig. 15. Comparison of using Adobe© Live Trace to vectorize directly on a rough sketch and on the output of our approach. Top row shows the input image and the output of our approach. Bottom row shows the vectorized results using Adobe© Live Trace. Image is copyrighted by David Revoy (www.davidrevoy.com) and licensed under CC-by 4.0.

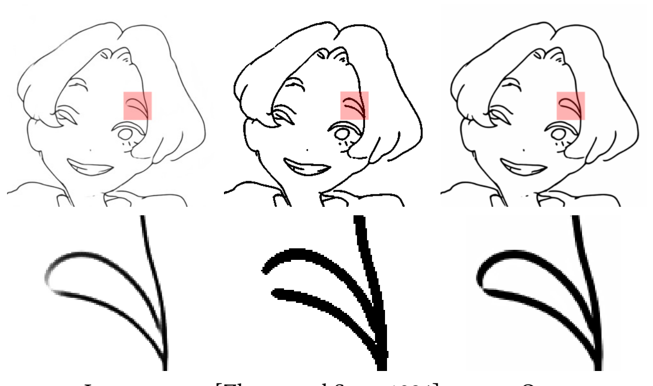


Fig. 16. Example of changing the line width of an image processed by our inking approach by making it 2 pixels wider. The area in red shown in the first row zoomed in on the second row.

detailed control of the output, professional users might not be completely satisfied with the results as it might not be expressive as being inked by hand. In this case, post-processing the result of our approach by hand with professional inking software, or if time permits, inking from scratch by hand, would allow for more minute

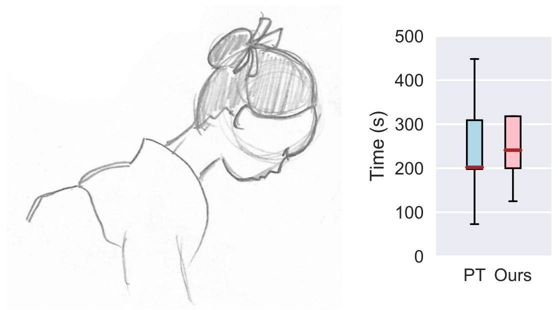


Fig. 17. Example of a rough sketch in which our approach takes roughly the same time as Professional Tools (PT) to ink, taken from our user study. On the right we show the timing results.

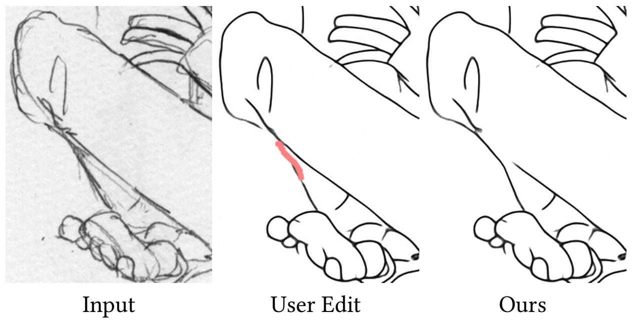


Fig. 18. Visualization of the non-local nature of the user edits.

control of the inking result. We believe a logical next-step would be tight integration with professional inking software, in order to provide the best of both worlds. In this setting, incorporating pressure sensitivity of the user input would also likely be beneficial.

ACKNOWLEDGMENTS

This work was partially supported by JST CREST Grant Number JPMJCR14D1, JST ACT-I Grant Number JPMJPR16UD, and JST ACT-I Grant Number JPMJPR16U3.

REFERENCES

Maher Ahmed and Rabab Ward. 2002. A rotation invariant rule-based thinning algorithm for character recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24, 12 (2002), 1672–1678.

Seok-Hyung Bae, Ravin Balakrishnan, and Karan Singh. 2008. ILoveSketch: As-natural-as-possible Sketching System for Creating 3D Curve Models. In *ACM Symposium on User Interface Software and Technology*. 151–160.

Houssem Chatbri and Keisuke Kameyama. 2014. Using scale space filtering to make thinning algorithms robust against noise in sketch images. *Pattern Recognition Letters* 42 (2014), 1–10.

Jiazhou Chen, Gaël Guennebaud, Pascal Barla, and Xavier Granier. 2013. Non-Oriented MLS Gradient Fields. *Computer Graphics Forum* 32, 8 (2013), 98–109.

David H Douglas and Thomas K Peucker. 1973. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization* 10, 2 (1973), 112–122.

Charles R Dyer and Azriel Rosenfeld. 1979. Thinning algorithms for gray-scale pictures. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 1 (1979), 88–89.

Mathias Eitz, James Hays, and Marc Alexa. 2012. How Do Humans Sketch Objects? *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 31, 4 (2012), 44:1–44:10.

Jean-Dominique Favreau, Florent Lafarge, and Adrien Bousseau. 2016. Fidelity vs. Simplicity: a Global Approach to Line Drawing Vectorization. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 35, 4 (2016).

Jakub Fišer, Paul Asente, Stephen Schiller, and Daniel Sýkora. 2015. ShipShape: A Drawing Beautification Assistant. In *Workshop on Sketch-Based Interfaces and Modeling*. 49–57.

Cindy Grimm and Pushkar Joshi. 2012. Just DrawIt: A 3D Sketching System. In *nternational Symposium on Sketch-Based Interfaces and Modeling*. 121–130.

David Ha and Douglas Eck. 2018. A Neural Representation of Sketch Drawings. In *International Conference on Learning Representations.*

Xavier Hilaire and Karl Tombre. 2006. Robust and accurate vectorization of line drawings. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28, 6 (2006), 890–904.

Takeo Igarashi, Satoshi Matsuoka, Sachiko Kawachiya, and Hidehiko Tanaka. 1997. Interactive Beautification: A Technique for Rapid Geometric Design. In *ACM Symposium on User Interface Software and Technology*. 105–114.

Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*.

Jang and Roland T. Chin. 1990. Analysis of thinning algorithms using mathematical morphology. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12, 6 (1990), 541–551.

Louisa Lam, Seong-Whan Lee, and Ching Y Suen. 1992. Thinning methodologies-a comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14, 9 (1992), 869–885.

David Lindlbauer, Michael Haller, Mark S. Hancock, Stacey D. Scott, and Wolfgang Stuerzlinger. 2013. Perceptual grouping: selection assistance for digital sketching. In *International Conference on Interactive Tabletops and Surfaces*. 51–60.

Xueting Liu, Tien-Tsin Wong, and Pheng-Ann Heng. 2015. Closure-aware Sketch Simplification. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 34, 6 (2015), 168:1–168:10.

Henry B Mann and Donald R Whitney. 1947. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics* (1947), 50–60.

Gary Martin, Steve Rude, and Terry Austin. 1997. *The Art of Comic Book Inking.* Dark Horse Comics.

Ana Maria Mendonca and Aurelio Campilho. 2006. Segmentation of retinal blood vessels by combining the detection of centerlines and morphological reconstruction. *IEEE Transactions on Medical Imaging* 25, 9 (2006), 1200–1213.

Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning*. 807–814.

Gioacchino Noris, Alexander Hornung, Robert W. Sumner, Maryann Simmons, and Markus Gross. 2013. Topology-driven Vectorization of Clean Line Drawings. *ACM Transactions on Graphics* 32, 1 (2013), 4:1–4:11.

Günay Orbay and Levent Burak Kara. 2011. Beautification of Design Sketches Using Trainable Stroke Clustering and Curve Fitting. *IEEE Transactions on Visualization and Computer Graphics* 17, 5 (2011), 694–708.

Urs Ramer. 1972. An iterative procedure for the polygonal approximation of plane curves. *Computer graphics and image processing* 1, 3 (1972), 244–256.

Patsorn Sangkloy, Jingwan Lu, Chen Fang, FIsher Yu, and James Hays. 2017. Scribbler: Controlling Deep Image Synthesis with Sketch and Color. In *IEEE Conference on Computer Vision and Pattern Recognition.*

Amit Shesh and Baoquan Chen. 2008. Efficient and Dynamic Simplification of Line Drawings. *Computer Graphics Forum* 27, 2 (2008), 537–545.

Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. 2016. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *IEEE Conference on Computer Vision and Pattern Recognition.*

Edgar Simo-Serra, Satoshi Iizuka, and Hiroshi Ishikawa. 2018. Mastering Sketching: Adversarial Augmentation for Structured Prediction. *ACM Transactions on Graphics* 37, 1 (2018).

Edgar Simo-Serra, Satoshi Iizuka, Kazuma Sasaki, and Hiroshi Ishikawa. 2016. Learning to Simplify: Fully Convolutional Networks for Rough Sketch Cleanup. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 35, 4 (2016).

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15 (2014), 1929–1958.

Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. 2015. Efficient object localization using convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition.*

Jun Xie, Holger Winnemöller, Wilmot Li, and Stephen Schiller. 2017. Interactive Vectorization. In *ACM CHI Conference on Human Factors in Computing Systems.*

Matthew D. Zeiler. 2012. ADADELTA: An Adaptive Learning Rate Method. *arXiv preprint arXiv:1212.5701* (2012).

Richard Zhang, Jun-Yan Zhu, Phillip Isola, Xinyang Geng, Angela S Lin, Tianhe Yu, and Alexei A Efros. 2017. Real-Time User-Guided Image Colorization with Learned Deep Priors. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 9, 4 (2017).

TY Zhang and Ching Y. Suen. 1984. A fast parallel algorithm for thinning digital patterns. *Commun. ACM* 27, 3 (1984), 236–239.