De-rendering Stylized Texts

Wataru Shimoda[1] Daichi Haraguchi[2] Seiichi Uchida[2] Kota Yamaguchi[1]

[1]CyberAgent [2]Kyushu University

Abstract

Editing raster text is a promising but challenging task. We propose to apply text vectorization for the task of raster text editing in display media, such as posters, web pages, or advertisements. In our approach, instead of applying image transformation or generation in the raster domain, we learn a text vectorization model to parse all the rendering parameters including text, location, size, font, style, effects, and hidden background, then utilize those parameters for reconstruction and any editing task. Our text vectorization takes advantage of differentiable text rendering to accurately reproduce the input raster text in a resolution-free parametric format. We show in the experiments that our approach can successfully parse text, styling, and background information in the unified model, and produces artifact-free text editing compared to a raster baseline.

1. Introduction

Typography is the art of visually arranging letters and text. Typography greatly affects how people perceive textual content in graphic design. Graphic designers carefully arrange and stylize text to convey their message in display media, such as posters, web pages, or advertisements. In computer vision, letters and texts have been predominantly the subject of optical character recognition (OCR), where the goal is to identify characters in the given image. This includes both OCR in printed documents or automatic scene text recognition for navigation. However, often, typographic information of text, such as font, outline, or shadow, is ignored in text recognition because the OCR does not target at reproducing typography in the output.

In this paper, we consider raster text editing as a text vectorization problem. There have been a few attempts at editing raster text mainly in scene images [32, 43, 45]. Previous work frames text editing as a style transfer problem in the pixel domain, where the goal is to apply the original style and effect in the input image to the target characters in the destination. However, the major limitation of the style transfer approach is that 1) it is hard to avoid artifacts in the resulting image, and 2) a pixel-based model is resolution-dependent. Our main target is display media. Compared to scene text images [32, 43, 45] that often involve external noise due to lighting condition, we often observe noise-free texts in display media which is prone to small artifacts on reconstruction. Editing in the vector format has a clear advantage in display media, as rendering results in consistent and sharp drawing at any resolution. Our approach is equivalent to viewing text recognition as a *de-rendering* problem [42], where the goal is to predict rendering parameters. Figure 1 illustrates our approach to the task of raster text editing.

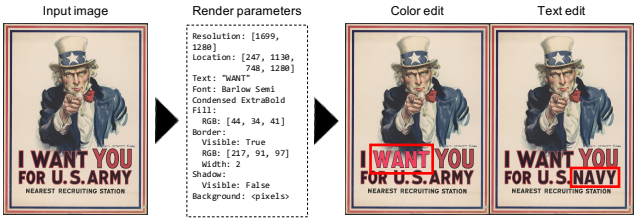

Figure 1. The proposed text editing approach. Once the rendering parameters are recovered, we can apply any text editing and style manipulation.

Text vectorization can be an ill-posed problem. For editing raster text in the vector format, we need to first parse characters, text styling information such as font and effects, and hidden background pixels. Once those rendering parameters are recovered from the input image, we can edit and render the target text using a rasterizer. Our approach hence has to solve three sub-problems: 1) OCR, 2) background inpainting, and 3) styling attribute recognition. OCR has a long history of research that dates back to the 1870s [34]. In parallel, inpainting has been studied in numerous literature [25, 50, 31, 24]. It might look straightforward to add styling attribute recognition to those two problems for our task. However, parsing various stylistic attributes is not trivial, as the presence of multiple rendering effects can easily lead to an ill-posed decomposition prob-

lem. For example, it is impossible to decompose the border and fill colors if two colors are the same. For solving such an ill-posed problem, we train neural networks to predict statistically plausible parameters.

Our model parses text, background, and styling attributes in two steps; we obtain the initial rendering parameters from feedforward inference, then refine the parameters by feedback inference. Our feedback refinement incorporates differentiable rendering to reproduce the given raster text, and fit the parameters to the raster input using reconstruction loss. Following SynthText [9], we train our feedforward model on text images we generate using a rendering engine. In the experiments, we show that our vectorization model accurately parses the text information. Further, we demonstrate that we can successfully edit texts by an off-the-shelf rendering engine using the parsed rendering parameters.

We summarize our contributions below.

- We formulate raster text editing as a *de-rendering* problem, where the task is to parse potentially ill-posed rendering parameters from the given image.
- We propose a vectorization model to parse detailed text information, using forward and backward inference procedure that takes advantage of differentiable rendering.
- We empirically show that our model achieves high quality parsing of rendering parameters. We also demonstrate that the parsed information can be successfully utilized in a 2D graphics engine for downstream edit tasks.

2. Related work

Optical character recognition OCR has a long history of research. The earliest ideas of OCR are conceived in the 1870s in Fournier d'Albe's Optophone and Tauschek's reading machine [34]. In the past, OCR models could not deal with font variations, but recent methods developed for scene texts can detect and recognize any fonts. Among many literature, EAST [53] and CRAFT [3] are well-known recent detection approaches. FOTS [26] and CharNet [44] report an end-to-end detection and recognition pipelines. FOTS incorporates a detection model and CRNN after ROI rotation, whereas CharNet utilizes a pixel-wise character classifier for assigning a character label to detect bounding boxes. In this paper, we adopt the CharNet model for the OCR module, but our model is not restricted to a specific OCR model.

Font recognition Compared to OCR, there has been little work for font recognition. Among a few literature, DeepFont [39] reports a deep learning approach to categorize fonts. The major challenge in font recognition is that fonts can be arbitrarily created by font designers, and it is often impossible to define a finite set of font categories. For this reason, some work instead resorts to font similarity evaluation [10, 5]. In this paper, we only use a fixed set of fonts to avoid the handling of infinite fonts.

Image vectorization Image vectorization aims at recognizing drawing entities such as lines or shape primitives from an image, and a classic example is line detection by Hough transform. Image vectorization has been actively studied and is still an open problem [13, 29, 7, 4, 18, 36, 1, 21, 33, 14]. Jia *et al*. [15] propose an approach to preserve an object arrangement and positional relationship using a depth map. Favreau *et al*. propose a method for vector clip-arts [8]. They decompose the input image into a small number of semi-transparent layers by color gradients. Liu *et al*. [23] report a method to convert a rasterized floor-plan into a vector format. Kim *et al*. [17] turn Chinese character images into vector formats by converting segmentation to overlapping strokes. Our approach shares image vectorization approaches in that text effects can be seen as image layers, but the target recognition entity is characters and rendering parameters rather than low-level shape primitives.

Raster text editing There has been a few recent work on raster text editing based on generative models in the pixel domain. STEFANN [32] proposes a model that generates a character mask for swapping and transferring color from the source character to the target, then paste the transferred character onto the background. SRNet [43] takes a raster text image and a target text in pixels, encodes them into a feature space, and generates a stylized target text as a decoding process. Yang *et al*. [45] further improve SRNet by manipulating geometric control points of characters to move text locations. Our work does not rely on pixel-domain generative models for text and styles but instead predicts parametric representation necessary for reproducing the raster text image.

Scene text inpainting Apart from editing text, some recent work solely focuses on erasing text by inpainting [30, 38, 52, 51, 22]. Nakamura *et al*. [30] are one of the first attempts that applies an inpainting approach to erase scene text. Zdeneck *et al*. consider text erasing under a weakly supervised setting [51]. The background inpainting module in our model considers the same problem, yet our model further considers foreground text in terms of rendering.

Text style transfer Font style transfer is yet another line of relevant work. Most of the work is based on generative networks except for Awesome Typography [46], where Yang *et al*. propose to transfer text effects by a patch matching algorithm. Azadi et al. [2] train the GAN-based model

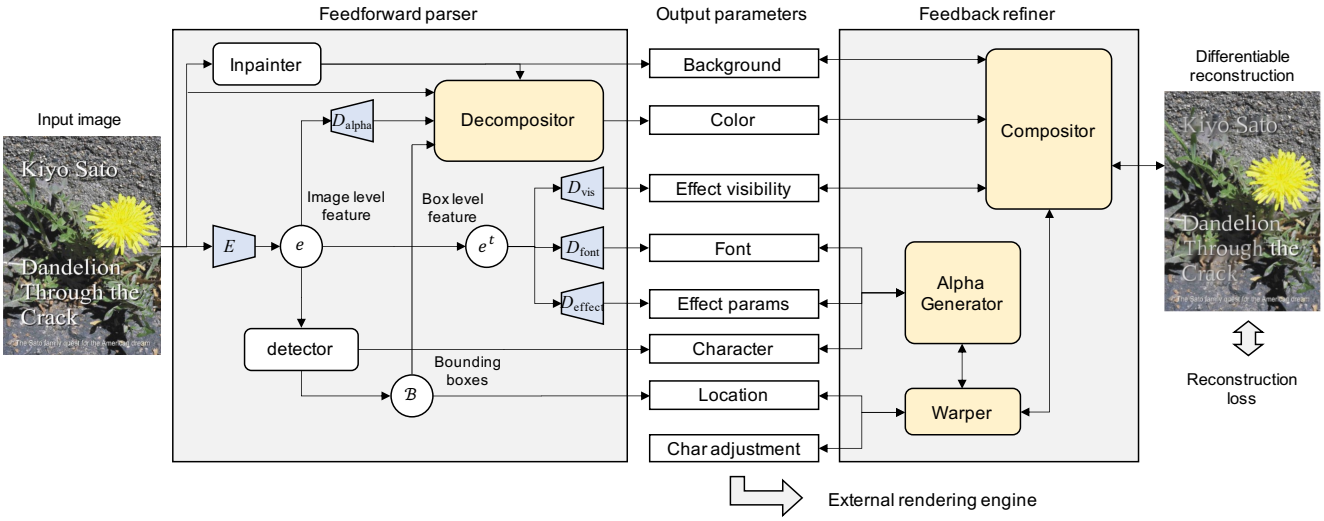

Figure 2. Overview of our approach. Our vectorization model parses rendering parameters from the raster text image in two stages. The feedforward inference gives the initial prediction of rendering parameters, and the feedback inference refines the solution by minimizing reconstruction loss via differentiable rendering.

for character style transfer. TETGAN [47] is another GAN-based character style transfer method that preserves various effects. Yang *et al*. stylize text with arbitrary texture [48, 49]. Wenjing *et al*. propose to transfer style for decorative elements [40]. Men *et al*. transfer video motion to text style [28].

Our work does not consider a highly-complex textured text image. Instead, we model basic text effects by parameters in the rendering engine including font, fill, border, and shadow, so that we can interpret and easily control the effect for editing. Although it is our future work to expand the effect kinds, typical typography in display media do not exhibit complex combination of effects.

3. Our approach

In this paper, we consider raster text editing as a text vectorization problem, where we solve three sub-problems: 1) OCR, 2) background inpainting, and 3) text attribute recognition. We approach this task by two-stage inference model consisting of a feedforward parser and a feedback refiner. We illustrate our model architecture in Fig 2.

3.1. Feedforward parsing

Our feedforward parser builds upon an OCR model that detects and recognizes characters in the image. We utilize an image-level feature map and box-level feature maps from the OCR model to further parse background colors, alpha maps of effects, and text attributes at each bounding box.

Detection and recognition Formally, our model takes a raster image x and first detects a set of bounding boxes $\mathcal{B}(x) = \{b^t, b^c\}$, where b^t and b^c are boundig boxes for texts and characters. For the detection task, we train an encoder E to obtain a global feature map $e(x) \in \mathbb{R}^{C\times H\times W}$, where C, H, W are the channels, height, width of the feature map. We use the hourglass-like network [44] to obtain the feature map. [1] The detector localizes the location of characters in the image. We follow the approach of CharNet [44] and generate a foreground segmentation and a geometry map that predicts distance to the center of the region to obtain localized bounding boxes $\mathcal{B}(x)$. After localizing text and character regions, we apply a character decoder $D_{\text{character}}$ to predict a sequence of characters. Internally, our model combines the foreground segmentation, the geometry map, and a character class map to parse the text in the given region. Note that we use CharNet [44] to implement our model, but can be any other model as long as characters can be localized.

Font recognition Similarly to text, we predict font categories of the given region using the local feature maps at the given bounding box $e^t_b(x)$ and $e^c_b(x)$, which is a subset of the global feature map $e(x)$ at the bounding box b^t and b^c. We apply pooling followed by a multi-layer perceptron to classify fonts.

Background inpainting The purpose of the background inpainting is to erase foreground raster text so that afterwards we can render manipulated text on the occluded background region. We can apply any image inpainting approach, but the challenge here is that we need to erase not

[1] We show the detailed architecture in the supplementary.

only text but also composited text effects. Since inpainting is a pixel-domain problem, we adopt a pixel-based generator. Our background inpainter takes a raster text image and predicts background RGB pixels x_{bg}. Note that we utilize OCR (Sec 3.1) and pixel-level alpha (Sec 3.1) to specify the region to inpaint. In our experiment, we adopt either an off-the-shelf inpainting method [37] or we learn the inpainting model of MEDFE [25].

Text effects Text effects involves a complex compositing process. We briefly describe the compositing process in the following. Formally, let us denote an alpha map of the k-th effect layer by $\alpha_k(i,j) = f_k(z(i,j), \theta_k)$, where f_k is an effect function that generates an alpha map given the source shape z and θ_k is the effect parameter. In our setting, z is a text rasterizer that generates a mask pixel at location (i,j). For a plain source-over blending on an opaque background[2], a rendering engine composites the k-th effect layer to the background image in the following equation:

$$\mathbf{c}(i,j) = \quad (1-\alpha_k(i,j))\mathbf{c}_{\mathrm{bg}}(i,j) + \alpha_k(i,j)\mathbf{c}_k, \qquad (1)$$

where $\mathbf{c}(i,j)$ is the color of the composited image, $\mathbf{c}_{\mathrm{bg}}(i,j)$ is the background color, and $\mathbf{c}_k$ is the effect color. When there are multiple effects, a rendering engine repeats the above compositing operation by substituting $\mathbf{c}_{\mathrm{bg}}(i,j) \leftarrow \mathbf{c}(i,j)$ at each effect k.

In this paper, we restrict the available effects to three basic but commonly-used kinds: *fill*, *border*, and *shadow*. We also assume the ordering of the effects is fixed to (shadow, fill, border). In our setting, all effects have color $\mathbf{c}_k$. Shadow and border effects have visibility parameter $v_k \in \{0,1\}$ that indicates whether the effect is enabled. Border has a width parameter: $\theta_{\mathrm{border}} = \{p_{\mathrm{width}}\}$ and shadow has three additional parameters that control the effect generation: $\theta_{\mathrm{shadow}} = \{p_{\mathrm{blur}}, p_{\mathrm{dx}}, p_{\mathrm{dy}}\}$. For each effect $k \in \{\mathrm{fill}, \mathrm{border}, \mathrm{shadow}\}$, we learn effect decoders from the feature maps:

$$\hat{v}_k = D_{v,k}(e_b(x)), \qquad (2)$$

$$\hat{\theta}_k = D_{\theta,k}(e_b(x)). \qquad (3)$$

For decoding color $\mathbf{c}_k$, we first decode the alpha map α_k for the entire canvas, and sequentially decomposite the color following the inverse of eq 1 and averaging the color in the bounding box region:

$$\hat{\alpha}_k = D_{\alpha,k}(e(x)), \qquad (4)$$

$$\mathbf{y}_{i,j,k} = \frac{\mathbf{c}(i,j) - (1-\hat{\alpha}_k(i,j))\mathbf{c}_{\mathrm{bg}}(i,j)}{\hat{\alpha}_k(i,j)}, \qquad (5)$$

$$\hat{\mathbf{c}}_k = \arg\max_{\mathbf{y}_{i,j,k}} P(\mathbf{y}_{i,j,k}), \qquad (6)$$

[2]Generic blending further considers different color blend modes, but we leave out for simplicity. See https://www.w3.org/TR/compositing-1/ for further details.

where the decompositing order is (border, fill, shadow), and if $\hat{v}_k = 0$, we skip decompositing of the effect k. For the initial border decompositing, we set $\mathbf{c}_b(i,j)$ from the result of the inpainting in Sec 3.1, and otherwise use the color of the previously decomposited effect.

3.2. Feedback refinement

While the feedforward parsing produces plausible rendering parameters, usually there are still inaccuracy in exactly reproducing the appearance of the input image. In our feedback refinement stage, we fill the appearance gap between the input and the vector reconstruction by minimizing the error in the rasterized image. The goal is to minimize the error between the input image x and the raster reconstruction $\mathcal{R}$ with all the rendering parameters Θ:

$$\min_{\Theta} |\mathcal{R}(\Theta) - x|. \qquad (7)$$

The raster reconstruction function $\mathcal{R}$ produces the final reconstruction using compositing equation (eq 1). As long as the compositing process is differentiable, we can apply back propagation to efficiently optimize parameters Θ. We propose to reconstruct a raster image using pre-rendered alpha maps so that the rendering process becomes differentiable. Note, while there exist recent approaches on differentiable rendering of strokes or shape primitives [19, 6, 16], we are not aware of any existing work on direct vectorization of text styling.

Pre-rendered alpha maps In the compositing process (eq 1), the glyph shape function z and the effect function f_k is usually not differentiable with respect parameters like font category and character class. We circumvent this issue by replacing the actual effect function by pre-rendering. The key idea is to rasterize non-differentiable glyphs and effects beforehand so that we can approximate rendering process as a linear combination of pre-rendered shapes. Let us denote the non-differentiable rendering parameters by $\Theta' = \{\theta_l | l \in \{\mathrm{char}, \mathrm{font}, \cdots\}\} \subset \Theta$. We approximate the non-differentiable shape function z by:

$$z(i,j;\Theta') = \sum_{\Theta'} P(\Theta') A_{\Theta'}(i,j), \qquad (8)$$

where $P(\Theta') = \prod_l P(\theta_l)$ is the joint probability (or so-called attention map) of parameter Θ', and $A_{\Theta'}(i,j)$ is the pre-rendered alpha maps for the corresponding parameter set Θ'. Since Θ' is defined over a finite set, we can pre-render $A_{\Theta'}(i,j)$ for all possible combination of Θ'. $P(\Theta')$ is differentiable with respect to Θ', and consequently eq 8 becomes also differentiable. Similarly, we approximate the borderline effect function f_k in the same pre-rendering approach to make differentiable approximation. An alpha map constructed from eq 8 visually looks blurred because the

shape is a linear combination of different font glyphs. However, the purpose of this alpha construction is to compute gradients over Θ' and not to produce the final raster reconstruction.

In this work, we use 100 fonts for the pre-rendered alpha maps randomly picked from Google Fonts[3]. Note that the pre-rendered alpha maps require large memory footprint if the parameter space is huge; e.g., if there are 100 fonts for 100 characters, we have to pre-render and keep 100×100 alpha maps, and also computing shape z becomes expensive. In practice, we compute sparse attention map $P(\Theta')$ for only the top 20 fonts and zero-out the rest to save computation. We choose 64×64 resolution for alpha maps considering the balance between the reconstruction quality and the memory footprint.

Character adjustment Although the OCR model gives the initial prediction of character bounding boxes, those locations do not perfectly align with the actual position of each character in the given input image. We introduce image warping to adjust the location of each character during feedback refinement. We prepare affine transformation parameters for each character and also for the word-level bounding box, and solve for the best adjustment during the optimization (eq 7). We set initial warping parameters to identity transformation at the beginning.

Differentiable compositing We use pre-rendered alpha for fill and border effects, but shadow has differentiable effect such as blur or offset. We directly implement those differentiable operators for shadow effect f_{shadow}.

To reconstruct an image, we composite alpha maps by eq 1 and image warping in character adjustment. The visibility flag controls whether to enable the effect compositing. All the operators are differentiable with respect to parameters Θ. We binalize the effect visibility v with differentiable binarization [20] and multiply to the foreground alpha. Note that our differentiable reconstruction is only for refinement purpose. The final output of our model is the refined rendering parameters Θ.

We show in Fig 3 the intermediate results during feedback refinement. Our refinement process starts from the rough initialization by feedforward parser, and gradually fits the parameters to reproduce the given raster image.

Refinement details We do not optimize location and background during refinement, because bounding box generation in our OCR model is not differentiable. Background pixels are differentiable, but we observe unstable behavior during optimization, perhaps due to ill-posed decompositing setup. We also do not optimize character probability

[3] https://github.com/google/fonts

Figure 3. Intermediate outputs during feedback refinement.

maps of OCR because of unstable behavior. We refine parameters using Adam optimizer with 200 iterations. In our implementation, optimization requires about 1 minute for one word.

3.3. Exporting rendering parameters

Parsed rendering parameters can be directly used in external 2D graphics rendering engine, though we need to adjust the scale of each parameter for the external format. Most of the parameters can be exported to the rendering engine with simple normalization, but we need to take care of computing geometric information such as font size or offset that our parser does not predict. Although our OCR model produces bounding boxes, it is not trivial to convert to those geometric information.

We convert these information in the following post-processing. We first transform character bounding boxes using affine parameters we obtain in letter adjustment. Next, we obtain a word-level bounding box by taking the minimum and maximum coordinates from the OCR bounding box and the set of character bounding boxes. Once we obtain both the character- and word-level bounding boxes, we use them to as guidance and conduct a grid search over configuration, starting from the parameters of best-fit character in the refinement.

4. Training

4.1. Training objective

For training our model, we minimize the following loss function, where each term corresponds to training of each decoder module in our model:

$$\mathcal{L} = \mathcal{L}_{\text{ocr}} + \mathcal{L}_{\text{inpaint}} + \lambda_{\text{font}}\mathcal{L}_{\text{font}} + \sum_k \lambda_{\text{alpha}}\mathcal{L}^k_{\text{alpha}} + \mathcal{L}^k_{\text{visibility}} + \mathcal{L}^k_{\text{param}}. \quad (9)$$

The loss functions for text recognition $\mathcal{L}_{\text{ocr}}$ and background inpainting $\mathcal{L}_{\text{inpaint}}$ are identical to those defined in CharNet [44] and MDEDF [25]. $\mathcal{L}_{\text{font}}$ is the categorical softmax cross entropy for font categories. $\mathcal{L}^k_{\text{alpha}}$ is the mean squared error of pixel-wise difference of the effect alpha map. $\mathcal{L}^k_{\text{visibility}}$ is the binary cross entropy of the effect visibility. $\mathcal{L}^k_{\text{param}}$ is the total of mean squared error for effect parameters. In the experiment, we apply weighting to each term for better convergence. λ_{font} and λ_{alpha} are the hyperparameters for balancing. We empirically set the hyperparameter values to $\lambda_{\text{font}} = 0.1$ and $\lambda_{\text{alpha}} = 10$.

4.2. Training data generator

We follow the approach of SynthText [9] to generate text images for training. In addition to the scene background images from SynthText, we collect background images from single-color background data, book cover dataset [12], BAM dataset [41] and FMD dataset [35]. For the book cover dataset and BAM dataset, we erase text regions by text detection and inpainting, then generate candidate locations from a saliency map. We take the vocabulary set from SynthText. We implement our data generator using the Skia graphics library[4], which can handle both font and effects in a purely resolution-free manner.

4.3. Implementation detail

We pre-train the OCR model for 5 epochs with 80,000 SynthText images. The condition is the same as the pre-training of CharNet [44]. The resolution of the input image is 640×640 , and the feature map is 160×160 with the channel size to be 256. We train our model with the SGD optimizer. We set the learning rate to 10^{-3} to the pre-trained encoder and 10^{-2} to all the other decoders, with batch size 4. We decrease the learning rate with cosine warm up [27].

We independently train the inpainting model on the generated text data [5]. In the training phase, we make holes for text regions with a mask generated by thresholding alpha values in each effect, and train to fill the holes. At test time, we make holes for text regions with predicted alpha maps, then fill the holes using the inpainting model. The resolution of the inpainting output [25] is 256×256. We resize the inpainted image to the target image size. This upsampling often causes undesired artifacts. To alleviate the quality issue, we replace text regions with text erased images and blend using predicted alpha.

Figure 4. Our two-stage vectorization process. From left to right, we show input images, reconstructions from the initial feedforward prediction, differentiable reconstructions of the feedback refinement, and the final vectorized images rendered using external graphics engine.

5. Experiments

In this experiments, we evaluate 1) how accurately our model parse text information in vectorized representation, and 2) how well our model can perform raster text editing through vectorized information. For experiments, we use two datasets, the book cover dataset [12] and the advertising image dataset [11].

[4] https://skia.org/

[5] We also attempted joint training in the experiment but did not observe a clear advantage.

Input images | Feedforward reconstructions | Refined reconstructions | Edit examples

Figure 5. Reconstruction and text editing examples from advertisement dataset. The leftmost column is the input, the second and the third columns are our differentiable reconstruction, and the last column is reconstructed and edited examples from the external graphic engine.

Input | SRNet | Ours | Style edited | Input | SRNet | Ours | Style edited

Figure 6. Text and style editing examples.

5.1. Qualitative results

We show in Fig 4 input images, initial reconstructions after feedforward parsing, reconstructions after feedback refinement, and final vectorization results using a 2D graphics engine from the book cover dataset. We observe that our approach successfully parses text information in various situations, including stylized texts with effects. In Fig 4, the first and second rows show examples of simple texts, and the third to fourth rows show examples of stylized texts with thin borderline and small shadow. The results demonstrate that our model can parse detailed effects parameters such as the weight of borderline and shadow orientation. The last two rows show challenging examples; the fifth row is an example of non-aligned text, and the last row is an example of unusual font. Our model parses non-aligned texts, but the vector reconstruction fails to render with the same style. When there is an unusual font, even though our reconstruction is inaccurate, our feedback refinement tries hard to adjust the choice of font and affine warping parameters to visually match the result.

We show reconstruction and text editing examples in advertisement datasets in Fig 5. Compared to book covers, advertisement images have various aspect ratios. Our approach successfully reconstructs complex text layouts in vector format with correct styling.

State of the art comparison We compare the quality of our editing approach with a state-of-the-art raster editing method SRNet [43]. SRNet takes a source style image and a target text mask to render the stylized target, where SR-

Table 1. Reconstruction performance.

Dataset	Method	L1 ↓	PSNR ↑
Book cover	Feedforward	0.15	29.22
	w/ refinement	0.08	30.02
	External rasterizer	0.18	29.73
Advertisement	Feedforward	0.16	28.77
	w/ refinement	0.09	29.37
	External rasterizer	0.19	29.16

Table 2. Component contribution in the book cover dataset.

Model	PSNR↑
Full refinement	30.02
w/o Color	29.53
w/o Font	29.78
w/o Shadow	29.79
w/o Border	29.99
No refinement	29.22

Net converts the text mask into a skeleton appearance and generates an image by transferring style from the reference style source. We use the alpha mask generated by the external rendering engine for the target text. Fig 6 shows qualitative comparisons between SRNet and our vectorization approach. We observe that SRNet often fails to draw a straight line and prone to be small artifacts, while our approach completely separates text shape from text effects in the parametric representation and does not suffer from artifacts. Note that our result has an additional benefit of easy and reproducible editing in the rendering parameters.

5.2. Quantitative evaluation

We quantitatively evaluate our model in terms of the reconstruction quality. We set up the following benchmark conditions.

Book cover We randomly pick 1,000 images from the book cover dataset, and apply an off-the-shelf OCR model [3] to extract title text region of the image for evaluation. In the book cover dataset, we use Telea's inpainting method [37] in our feedforward parser, because we find the off-the-shelf method already works well for book cover images.

Advertisement We randomly pick 508 images from the ad dataset and evaluate reconstruction performance on all text regions. We use the learned inpainting model [25].

To measure the reconstruction performance, we use the average L1 loss and peak signal-to-noise ratio (PSNR) between the input image and the reconstructed image. We compare the quality of 1) the differentiable reconstruction after feedforward parsing, 2) after feedback refinement, and 3) the final reconstruction in the external graphic engine.

We summarize the evaluation results in Table 1. We find that our two-stage vectorization approach effectively improves the reconstruction quality, as the refinement consistently improves both L1 and PSNR compared to the feedforward parser only. The final reconstruction using the external rasterizer does not necessarily gives the identical result to our differentiable reconstruction. Our refinement process includes approximation and also the exporting process (Sec 3.3) can introduce errors. Nevertheless, we find the reconstruction quality reasonably well, and PSNR is consistently better than the initial feedforward reconstruction. Filling the gap between the differentiable reconstruction and the exported result is our future work.

Component analysis We show in Table 2 how PSNR changes as we skip feedback refinement in rendering parameters in the book cover dataset. We skip *color*, *font*, *shadow*, and *border* effects in this ablation study. As the result suggest, effect components that has larger area in the resulting appearance have more performance influence on the reconstruction quality; incorrect *color* most negatively affect the resulting visual quality, while other effects occupy smaller regions and therefore quantitatively have less minor impact on PSNR.

5.3. Discussion

One drawback of our approach is that we cannot handle arbitrary styles that are not defined in our basic effect models or unusual glyph shapes. We also do not consider detailed rendering parameters such as kerning or geometric transformation. It is our future work to extend the available fonts or effects, where we have to address efficient differentiable rendering for a large combination of non-differentiable operations (Sec 3.2). However, we note that our basic effects already cover the majority of real-world typography in display media.

6. Conclusion

We present a vectorization approach to edit raster text images. Our approach inpaints the background and accurately parses foreground text information for reconstruction in an external rendering engine. We show in the experiments our model successfully reproduces the given raster text in vector graphic format. Our vectorization approach has a clear advantage in resolution-free editing, especially for display media. In the future, we wish to expand the available effects and blending operations. One limitation of our approach is that effects must be pre-defined by the rendering engine. We wish to expand the available effects and blending operations in the future.

References

[1] O. Alexandrina, B. Adrien, W. Holger, B. Pascal, T. Joëlle, and S. David. Diffusion curves: A vector representation for smooth-shaded images. In *SIGGRAPH*, 2008. 2

[2] S. Azadi, M. Fisher, V. Kim, Z. Wang, E. Shechtman, and T. Darrell. Multi-content gan for few-shot font style transfer. In *CVPR*, 2018. 2

[3] Y. Baek, B. Lee, D. Han, S. Yun, and H. Lee. Character region awareness for text detection. In *CVPR*, 2019. 2, 8

[4] A. Carlier, M. Danelljan, A. Alahi, and R. Timofte. DeepSVG: A hierarchical generative network for vector graphics animation. In *NeurIPS*, 2020. 2

[5] T. Chen, Z. Wang, N. Xu, H. Jin, and J. Luo. Large-scale tag-based font retrieval with generative feature learning. In *ICCV*, 2019. 2

[6] Ayan. Das, Yongxin. Yang, Timothy. Hospedales, Tao. Xiang, and Yi-Zhe. Song. B´ezierSketch: A generative model for scalable vector sketches. *ECCV*, 2020. 4

[7] V. Egiazarian, O. Voynov, A. Artemov, D. Volkhonskiy, A. Safin, M. Taktasheva, D. Zorin, and E. Burnaev. Deep vectorization of technical drawings. In *ECCV*, 2020. 2

[8] J. Favreau, F. Lafarge, and A. Bousseau. Photo2clipart: Image abstraction and vectorization using layered linear gradients. *SIGGRAPH Asia*, 2017. 2

[9] A. Gupta, A. Vedaldi, and A. Zisserman. Synthtext in the wild dataset. *CVPR*, 2016. 2, 6

[10] D. Haraguchi, S. Harada, B. Iwana, Y. Shinahara, and S. Uchida. Character-independent font identification. In *IAPR International Workshop on Document Analysis Systems (DAS)*, 2019. 2

[11] Z. Hussain, M. Zhang, X. Zhang, K. Ye, C. Thomas, Z. Agha, N. Ong, and A. Kovashka. Automatic understanding of image and video advertisements. In *CVPR*, 2017. 6

[12] Brian Kenji Iwana, Syed Tahseen Raza Rizvi, Sheraz Ahmed, Andreas Dengel, and Seiichi Uchida. Judging a book by its cover. *arXiv preprint arXiv:1610.09204*, 2016. 6

[13] F. Jean-Dominique, L. Florent, and B. Adrien. Fidelity vs. simplicity: a global approach to line drawing vectorization. *SIGGRAPH*, 2016. 2

[14] S. Jeschke, D. Cline, and P. Wonka. Estimating color and texture parameters for vector graphics. *Computer Graphics Forum*, 2011. 2

[15] Z. Jia, A. Gallagher, Y. Chang, and T. Chen. A learning-based framework for depth ordering. In *CVPR*, 2012. 2

[16] Byungsoo Kim, Oliver Wang, A. Cengiz Öztireli, and Markus Gross. Semantic segmentation for line drawing vectorization using neural networks. *Eurographics*, 2018. 4

[17] B. Kim, O. Wang, and M. Öztireli, C. Gross. Semantic segmentation for line drawing vectorization using neural networks. In *Eurographics*, 2018. 2

[18] G. Lecot and B. Levy. Ardeco: Automatic region detection and conversion. In *Symposium on Rendering*, 2006. 2

[19] Tzu-Mao Li, Michal Lukáč, Gharbi Michaël, and Jonathan Ragan-Kelley. Differentiable vector graphics rasterization for editing and learning. *SIGGRAPH Asia*, 39(6):193:1–193:15, 2020. 4

[20] M. Liao, Z. Wan, C. Yao, K. Chen, and X. Bai. Real-time scene text detection with differentiable binarization. In *AAAI*, 2020. 5

[21] Z. Liao, H. Hoppe, F. David, and Y. Yu. A subdivision-based representation for vector image editing. *IEEE Transactions on Visualization and Computer Graphics*, 2012. 2

[22] Chongyu Liu, Yuliang Liu, Lianwen Jin, Shuaitao Zhang, Canjie Luo, and Yongpan Wang. Erasenet: End-to-end text removal in the wild. *IEEE Transactions on Image Processing*, 29:8760–8775, 2020. 2

[23] C. Liu, J. Wu, P. Kohli, and Y. Furukawa. Raster-to-vector: Revisiting floorplan transformation. In *ICCV*, 2017. 2

[24] Guilin Liu, Fitsum A Reda, Kevin J Shih, Ting-Chun Wang, Andrew Tao, and Bryan Catanzaro. Image inpainting for irregular holes using partial convolutions. In *ECCV*, pages 85–100, 2018. 1

[25] H. Liu, B. Jiang, Y. Song, W. Huang, and C. Yang. Rethinking image inpainting via a mutual encoder-decoder with feature equalizations. In *ECCV*, 2020. 1, 4, 6, 8

[26] X. Liu, D. Liang, S. Yan, D. Chen, Y. Qiao, and J. Yan. Fots: Fast oriented text spotting with a unified network. *CVPR*, 2018. 2

[27] L. Loshchilov and F. Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *ICLR*, 2017. 6

[28] Y. Men, Z. Lian, Y. Tang, and J. Xiao. Dyntypo: Example-based dynamic text effects transfer. In *CVPR*, 2019. 3

[29] B. Mikhail and S. Justin. Vectorization of line drawings via polyvector fields. *SIGGRAPH*, 2019. 2

[30] Toshiki Nakamura, Anna Zhu, Keiji Yanai, and Seiichi Uchida. Scene text eraser. In *ICDAR*, volume 1, pages 832–837. IEEE, 2017. 2

[31] Kamyar Nazeri, Eric Ng, Tony Joseph, Faisal Qureshi, and Mehran Ebrahimi. Edgeconnect: Structure guided image inpainting using edge prediction. In *ICCV Workshops*, 2019. 1

[32] R. Prasun, B. Saumik, G. Subhankar, and P. Umapada. Stefann: Scene text editor using font adaptive neural network. In *CVPR*, 2020. 1, 2

[33] C. Richardt, J. Lopez-Moreno, A. Bousseau, M. Agrawala, and G. Drettakis. Vectorising bitmaps into semi-transparent gradient layers. *Computer Graphics Forum*, 2014. 2

[34] Herbert F Schantz. *History of OCR, optical character recognition*. Recognition Technologies Users Association, 1982. 1, 2

[35] Lavanya Sharan, Ruth Rosenholtz, and Edward Adelson. Material perception: What can you see in a brief glance? *Journal of Vision*, 9(8):784–784, 2009. 6

[36] Jian Sun, Lin Liang, Fang Wen, and Heung-Yeung Shum. Image vectorization using optimized gradient meshes. *ACM Transactions on Graphics (TOG)*, 26(3):11–es, 2007. 2

[37] Alexandru Telea. An image inpainting technique based on the fast marching method. *Journal of graphics tools*, 9(1):23–34, 2004. 4, 8

[38] Osman Tursun, Rui Zeng, Simon Denman, Sabesan Sivapalan, Sridha Sridharan, and Clinton Fookes. MTRNet: A generic scene text eraser. In *ICDAR*, pages 39–44. IEEE, 2019. 2

[39] Z. Wang, J. Yang, H. Jin, E. Shechtman, A. Agarwala, J. Brandt, and T. Huang. Deepfont: Identify your font from an image. *ACM Multimedia*, 2015. 2

[40] W. Wenjing, L. Jiaying, Y. Shuai, and G. Zongming. Typography with decor: Intelligent text style transfer. In *CVPR*, 2019. 3

[41] Michael J. Wilber, Chen Fang, Hailin Jin, Aaron Hertzmann, John Collomosse, and Serge Belongie. Bam! the behance artistic media dataset for recognition beyond photography. In *ICCV*, Oct 2017. 6

[42] J. Wu, J.B. Tenenbaum, and P. Kohli. Neural scene derendering. In *CVPR*, 2017. 1

[43] L. Wu, C. Zhang, J. Liu, J. Han, J. Liu, E. Ding, and X. Bai. Editing text in the wild. In *ACM Multimedia*, 2018. 1, 2, 7

[44] L. Xing, Z. Tian, W. Huang, and M. Scott. Convolutional character networks. *ICCV*, 2019. 2, 3, 6

[45] Q. Yang, HJ. Jin, J. Huang, and W. Lin. Swaptext: Image based texts transfer in scenes. In *CVPR*, 2020. 1, 2

[46] S. Yang, J. Liu, Z. Lian, and Z. Guo. Awesome typography: Statistics-based text effects transfer. In *CVPR*, 2017. 2

[47] S. Yang, J. Liu, W. Wang, and Z. Guo. Tet-gan: Text effects transfer via stylization and destylization. In *AAAI*, 2019. 3

[48] S. Yang, J. Liu, W. Yang, and Z. Guo. Context-aware text-based binary image stylization and synthesis. In *IEEE Trans. on Image Processing*, 2018. 3

[49] S. Yang, Z. Wang, Z. Wang, N. Xu, J. Liu, and Z. Guo. Controllable artistic text style transfer via shape-matching gan. In *ICCV*, 2019. 3

[50] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. Free-form image inpainting with gated convolution. In *ICCV*, 2019. 1

[51] Jan Zdenek and Hideki Nakayama. Erasing scene text with weak supervision. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 2238–2246, 2020. 2

[52] Shuaitao Zhang, Yuliang Liu, Lianwen Jin, Yaoxiong Huang, and Songxuan Lai. Ensnet: Ensconce text in the wild. In *AAAI*, volume 33, pages 801–808, 2019. 2

[53] X. Zhou, C. Yao, H. Wen, Y. Wang, S. Zhou, W. He, and J. Liang. East: An efficient and accurate scene text detector. In *CVPR*, 2017. 2