Segmentation-guided Layer-wise Image Vectorization with Gradient Fills

Hengyu Zhou, Hui Zhang✉, and Bin Wang✉

School of Software, Tsinghua University, P. R. China
zhouhy22@mails.tsinghua.edu.cn, {huizhang, wangbins}@tsinghua.edu.cn

Abstract. The widespread use of vector graphics creates a significant demand for vectorization methods. While recent learning-based techniques have shown their capability to create vector images of clear topology, filling these primitives with gradients remains a challenge. In this paper, we propose a segmentation-guided vectorization framework to convert raster images into concise vector graphics with radial gradient fills. With the guidance of an embedded gradient-aware segmentation subroutine, our approach progressively appends gradient-filled Bézier paths to the output, where primitive parameters are initiated with our newly designed initialization technique and are optimized to minimize our novel loss function. We build our method on a differentiable renderer with traditional segmentation algorithms to develop it as a model-free tool for raster-to-vector conversion. It is tested on various inputs to demonstrate its feasibility, independent of datasets, to synthesize vector graphics with improved visual quality and layer-wise topology compared to prior work.

Keywords: Vectorization · Segmentation · Differentiable rendering

1 Introduction

Vector graphics offer great flexibility in digital design as they can be easily edited and arbitrarily scaled. Vectorization, the procedure of converting raster images to vector ones, serves as a second way to creating vector graphics other than designing from scratch where extensive artistic skills are required, and also as a bridge between the rapidly developing raster image generation [27] and the relatively less studied vector generation [3, 8, 10, 24, 32].

Vectorization has been explored for decades with various representations proposed to divide images into non-overlapping regions [17, 29, 33, 35, 36]. Despite their capability of generating vivid vectorization of realistic images, the complex primitives and lack of hierarchy make the vector output less intuitive for manipulation.

Recent learning-based methods show their potential in preserving image hierarchy, where a vector image is often considered as a list of primitives and their order indicates the structure of the input. Many deep learning approaches

✉ Corresponding authors.

have been proposed to vectorize simple inputs [24] or images of a specific field of interest [6, 18, 25, 28], but their dependency on models limits them to a particular domain and are not trivially generalizable. LIVE [20], in contrast, utilizes DiffVG [16], a differentiable renderer, to present a model-free vectorization framework. It progressively translates a raster image into an SVG in a layer-wise hierarchy, through which the topology of the input is preserved within the order of geometric primitives. However, its lack of support for gradients results in excessive primitives being added in case of images with rich gradient effects, and adding such support is not as simple as replacing RGBA colors with gradient parameters, as elaborated in Sec. 3.4.

In this paper, we propose a novel segmentation-guided vectorization framework that extends the capability of LIVE to support radial gradients. The additional parameters of a radial gradient pose an increased challenge to optimization, where an effective method to determine whether a pixel contributes to a path's gradient fill is necessary. The key insight behind our idea is the similarity between finding contributing pixels and segmentation tasks, based on which we designed a segmentation-guided initialization procedure to progressively append new shapes to the vector output, and a novel loss function to optimize their geometric and gradient parameters. We evaluated our framework on several datasets with quantitative metrics and a user study, to show its effectiveness and superior performance compared to previous work.

To summarize our contributions:

- We introduce a segmentation-guided vectorization framework to create vector graphics automatically with layer-wise hierarchy and radial gradients.
- We propose a gradient-aware segmentation method to evaluate the pixel-wise contribution to the geometric and gradient parameters of a path.
- We take the segmentation as guidance for our new initialization technique and as a part of our novel segmentation-guided loss.

2 Related Work

2.1 Image Vectorization

Most traditional vectorization methods aim to create vector images of high fidelity with different representations, which could be roughly categorized into mesh-based ones and curve-based ones. The former representations divide the input image into non-overlapping 2D patches across which colors are interpolated [30]. Shapes of patches include triangular [17, 33, 36], rectangular [1, 29] or irregular ones such as bézigons [35]. The different selection of mesh shapes determines how patches are organized and how colors are interpolated within patches. The curved-based representations decompose the image into curves. Diffusion curves, for instance, use curves as geometric primitives with colors defined on sides of the curves [22, 34]. While these methods can yield near-photo-realistic results, they may fall short when it comes to ease of editing due to the complex primitives and loss of topological information, compared to a plain list of

simple primitives like how an SVG organizes a vector image. Recent learning-based vectorization work mainly takes the list-of-primitive approach, where the ordered primitives are seen as a sequence of drawing operations, and sequential prediction methods are applied to synthesize vector graphics. Models are used including recurrent neural networks [10,24], transformers [3], and are often combined with variational autoencoders [10,15,19,24]. DiffVG [16] as a differentiable renderer fills the gap between raster and vector graphics, with which loss functions on raster images could be used directly to optimize the vector images. A vectorization method optimizing all paths at once is also proposed in the work.

2.2 Image Topology

When synthesizing vector graphics, both similarity to the original image and correct topology are important for the ease of subsequent human manipulation. A related problem is layer decomposition, where images are decomposed into semi-transparent layers. With these layers being vector paths, such a method could serve as a layered vectorization approach. An interactive method [26] is among the first proposed to convert bitmaps into layered vector graphics. Photo2ClipArt [7] and other similar work [5] replace the heavy manual interaction with a user-provided segmentation input, from which the decomposition could be automatically determined. However, these methods require a concise segmentation for efficient and effective vectorization into a relatively small number of paths, where automatic segmentation algorithms struggle to serve the purpose [7].

Many learning-based methods formulate vector synthesis as sequential prediction problems, through which the order of primitives is naturally preserved. Some methods predict the primitive parameters directly. For example, Egiazarian et al. [6] use a transformer-based network to directly generate the vectorization of technical line drawings; MARVEL [28] applies deep reinforcement learning to predict strokes in black-and-white comics. Meanwhile, some approaches use variational autoencoders (VAE) [15] to create vector graphics. SketchRNN [10] is the first to introduce a Long Short-Term Memory (LSTM) [12] based VAE to the representation of vector sketches. The later DeepSVG [3] proposed a transformer-based VAE network that also encodes a vector image into a latent code, which could be decoded to reconstruct the vector paths. SVG-VAE [19] is the first to use raster images as the input to its encoder, thus it could be used to vectorize images. Im2Vec [24] also takes raster images as its input but drops the need for vector supervision with the help of DiffVG [16]. While these methods show great potential in creating topology-aware vector graphics, they rely on specific training datasets and may overlook the fine details in a complex input. The most similar work to ours is LIVE [20], in which a progressive framework is proposed to convert raster images into layered vector paths utilizing DiffVG for optimization, but its missing support for color gradients is not trivially achievable by simply optimizing gradient parameters. Our novel segmentation-guided method takes a step further to handle gradient effects properly, at the same level of simplicity as LIVE, requiring neither additional user input nor deep models.

3 Method

3.1 Method Overview

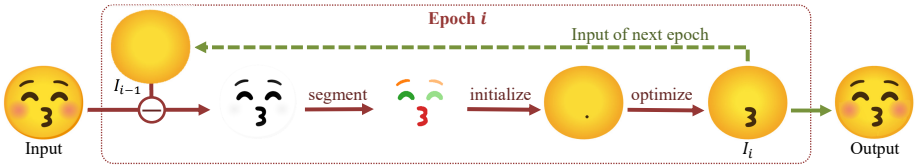


Fig. 1: Overview of our framework. *U+1F61A* from Noto Emoji [9] is used for demonstration.

We present a new framework to generate SVG figures from rasterized images, with support for radial gradient fills via a segmentation-guided initialization and optimization process.

Our framework works progressively, where at each epoch, single or multiple Bézier paths are added and optimized. Fig. 1 provides an overview of our method. At the beginning of each epoch i, we calculate the difference between the input raster image and the output I_{i-1} from the previous epoch. We segment it with our gradient-aware segmentation (Sec. 3.2), from which n_i segmented regions are selected for initialization of new paths (Sec. 3.3). All added paths, including those added in previous epochs, are optimized to minimize the vectorization loss (Sec. 3.4). The parameters to be optimized consist of geometric parameters including positions of curve control points and gradient parameters including their center, radius, and color stops. n_i, or the number of paths to be appended at i-th epoch is specified by the user along with the number of iteration epochs.

3.2 Gradient-aware Segmentation

(a) Input **(b)** LIVE **(c)** LIVE-30 **(d)** Mean shift **(e)** Ours

Fig. 2: Comparisons on segmentation methods. (a) is the raster input. (b) is by LIVE under default settings clustering colors into 200 bins. (c) is also by LIVE but with the number of bins set to 30. (d) is segmented using the Mean-shift [4] algorithm. (e) is by our method.

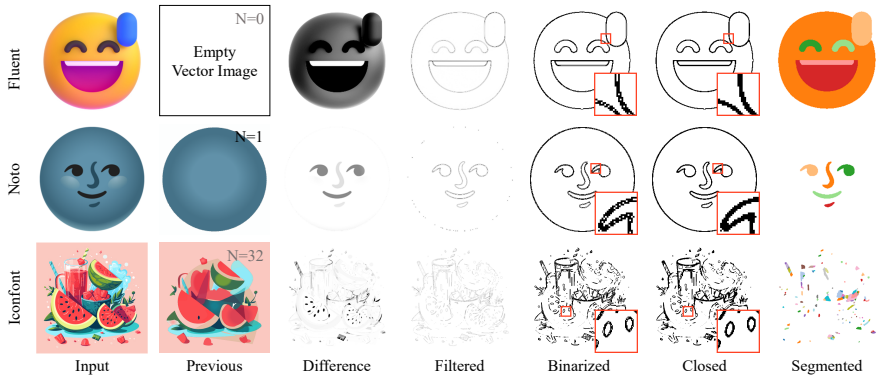


Fig. 3: Step-by-step outputs of our gradient-aware segmentation

Quality of path initialization is the key to effective vectorization, hence we present a gradient-aware segmentation method to better estimate the area that a path is likely to span over. Its result is used in the subsequent operations including initialization and optimization (Sec. 3.3, Sec. 3.4).

When paths are filled with a solid color, a connected component of similar or identical colors has a good chance of being covered by one path. LIVE [20] designed a component-wise initialization method, where pixels are clustered into buckets based on their l2-length over RGB channels, and connected pixels in the same bucket are considered one component. As gradients are considered, the clustering algorithm used by LIVE may result in an excessive segmentation, as shown by Fig. 2. Other clustering methods including Mean-shift [4] also suffer from over-segmentation for the same reason that colors within a path may vary more than colors between paths.

With the limitations of previous methods, our approach is designed to detect edges of gradients. Colors should derive smoothly inside a region filled with the same gradient fill and are likely to change abruptly at its boundary. Thus, we calculate the secondary spatial gradient with a discrete Laplacian filter $L = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ to identify such rapid changes at gradient boundaries. Our segmentation method features the following steps:

- The cross-correlation $S_0 \in \mathbb{R}_{w \times h \times 3}$ between the difference to the input and the 2D Laplacian filter is calculated as

$$S_0 = \text{correlate}((I - \hat{I})\mathbf{1}_{||\hat{I}-I||_2 > \epsilon}, L)\,,$$

  where $\hat{I}$ is the target and I is the synthesized image. Pixels with an error below $\epsilon = 0.1$ are excluded.
- S_0 is then summed over its RGB channels for a grayscale image $S_1 \in \mathbb{R}_{w \times h}$, where each pixel of S_1 has a value of $(S_1)_{ij} = \sum_{c=1}^{3} |(S_0)_{ijc}|$.
- The grayscale image S_1 is converted to a binary image $S_2 = \mathbf{1}_{S_1 > \text{Otsu}(S_1)}$ with the threshold determined via Otsu's method [23].

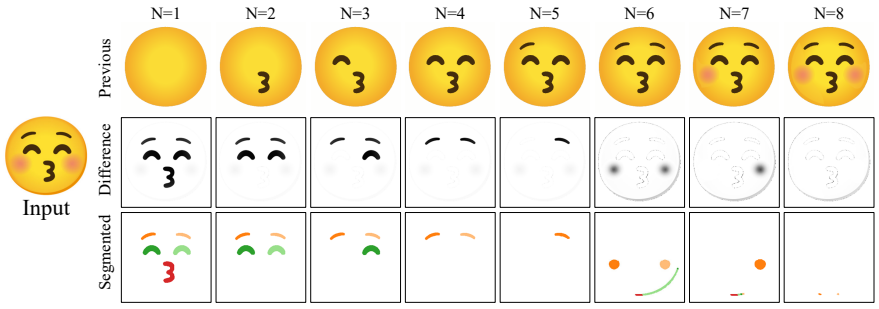


Fig. 4: With paths being progressively added, the overall difference between the current output and the raster input decreases, thus we apply Otsu's method [23] to automatically determine the threshold for binarization. In this example, the red cheeks get segmented after more significant differences including eyes and brows are fitted, thanks to the dynamic threshold. The differences are normalized and mapped to grayscale.

– Morphological closing [11] and the watershed algorithm [2] are then applied to S_2 for the final segmentation $S \in \mathbb{N}_{h \times w}$.

In Fig. 3 we showcase some intermediate results after each step. Since we are segmenting the difference between the output and the target, the already fitted pixels are ignored and the automatically determined threshold via Otsu's method decreases in response to the descending overall difference, as shown in Fig. 4. Compared to a fixed threshold, our method makes fewer assumptions about the input and avoids hyperparameters.

3.3 Segmentation-guided Initialization

Our vectorization method works in a progressive manner, throughout which we add one or more paths at each epoch. These newly added paths are initialized using our segmentation S as the guidance. For each path to be added, we select the segmentation region with the largest accumulated square error calculated as

$$w_i = \sum_{p \in \tilde{S}[i]} ||I_p - \hat{I}_p||^2 \tag{1}$$

where p is iterated over all pixels in i-th region. This approach prioritizes larger regions to encourage a hierarchical initialization order and prevents regions that are almost properly filled from being chosen. A circle path of four cubic Bézier curves [20] is added at the selected region's center of mass p_m. We fill the path with a radial gradient, which centers at p_m, with a diameter equal to the geometric mean of the width and the height of the region's bounding box clipped to $[0.2, 1.0]$. The two stop colors at offsets 0% and 100% are both initialized to the color of the input image I at p_m.

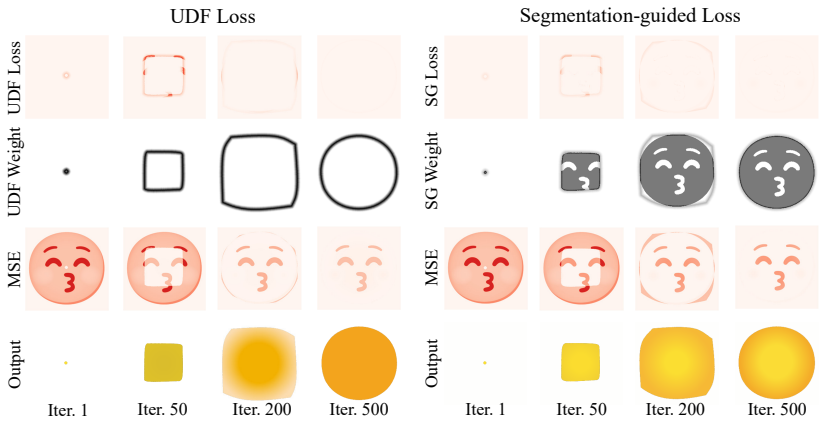


Fig. 5: Optimizing course for our method in comparison with UDF loss from LIVE. With our proposed segmentation-guided weight (SG Weight), the gradient fill is optimized to minimize the color error not only on the contour as UDF weight focuses on but also on colors inside the path, while excluding pixels occluded by eyes, brows, and the mouth.

3.4 Loss Function with Segmentation-guided weight

A common approach to measuring the difference between the synthesized image $\hat{I}$ and the target image I is through mean squared error (MSE). However, simply minimizing MSE results in colors biasing towards the average color inside a path, as suggested by LIVE [20]. LIVE tackles the problem with its designed UDF loss to focus on pixels on the contour, but for radial gradients, correct colors on the contour do not mean correctness inside, as indicated by Fig. 5. When gradients are optimized with UDF loss, colors at the center are not fitted well, since those pixels are ignored by the UDF weight. We draw from the insight by LIVE emphasizing the significance of pixels on edges and extend this concept to include all pixels within a path, except for those occluded by other paths, which is well estimated through our gradient-aware segmentation.

We formulate our segmentation-guided loss with this estimation of unoccluded pixels within a path. For each added path p_i, given the set of pixels covered by the path and the set of pixels within the segment from which the path is initialized, we take their intersection and mark these pixels as being focused. The union of all focused pixels forms a set F. We define the weight w_{SG} as:

$$w_{\text{SG}}(i) = \begin{cases} \max(d'_i, \alpha_s), & i \in F \\ d'_i(1-\alpha_s), & \text{otherwise} \end{cases}, \tag{2}$$

where i is an index of pixel and d'_i is the UDF weight from [20]. The UDF weight is introduced to give the pixels near the path contour a higher weight, as depicted in Fig. 5. α_s balances between the UDF weight d'_i and our segmentation weight,

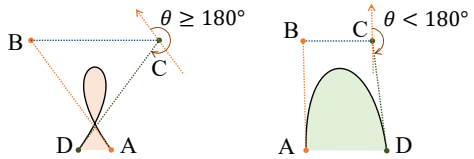


Fig. 6: Bézier paths with four control points

and is set to 0.6 empirically. d'_i is defined with:

$$d'_i = \frac{\mathrm{ReLU}(\tau - |d_i|)}{\sum_{j=1}^{w \times h} \mathrm{ReLU}(\tau - |d_j|)}, \tag{3}$$

where d_i is the distance from the pixel i to the nearest path contour, which is then normalized and saturated with threshold $\tau = 10$.

We re-weight the pixel-wise color error with our proposed weight w_{SG} for the final segmentation-guided loss:

$$\mathcal{L}_{\mathrm{SG}} = \frac{1}{3} \sum_{i=1}^{w \times h} w_{\mathrm{SG}}(i) \sum_{c=1}^{3} (I_{c,i} - \hat{I}_{c,i})^2, \tag{4}$$

where i is iterated over all pixels and squared color differences are averaged over the RGB channels.

We also introduce Xing loss [20] to help relieve self-intersection. Given a cubic Bézier curve with four control points A, B, C, and D, self-intersection is more likely to occur when the angle between $\boldsymbol{AB}$ and $\boldsymbol{CD}$ is greater than 180°, as shown in Fig. 6. The Xing loss $\mathcal{L}_{\mathrm{Xing}}$ is defined to penalize angles over 180°:

$$D_1 = \begin{cases} 1, \, \boldsymbol{AB} \times \boldsymbol{BC} > 0 \\ 0, \, otherwise \end{cases}, \; D_2 = \frac{\boldsymbol{AB} \times \boldsymbol{CD}}{\|\boldsymbol{AB}\| \, \|\boldsymbol{CD}\|} \tag{5}$$

$$\mathcal{L}_{\mathrm{Xing}} = D_1(\mathrm{ReLU}(-D_2)) + (1 - D_1)(\mathrm{ReLU}(D_2)). \tag{6}$$

Our optimization objective is defined as follows:

$$\mathcal{L} = \mathcal{L}_{\mathrm{SG}} + \lambda \mathcal{L}_{\mathrm{Xing}}, \tag{7}$$

where λ is set to 0.05 empirically.

4 Experiments

4.1 Implementation Details

We implement our framework with DiffVG [16] renderer based on PyTorch. Adam optimizer [14] is used with a learning rate of 10^{-2} and 1 for gradient parameters and path points respectively. We use scikit-image [31] for its implementation of morphological operations and the watershed algorithm [2].

Fig. 7: Exemplars from our chosen datasets for evaluation

4.2 Datasets

Noto Emoji We randomly select 256 emojis from Noto Emoji [9](Unicode 15.0) varying in colors and genres. In contrast to previous work [20,24], we take a more recent version of the dataset, consisting of resigned emojis filled with gradients instead of single colors. Emojis from this dataset consist of a relatively small number of paths with a clear hierarchy, thus we evaluate on this dataset to show our method's capability to decompose images into layers with gradients.

Fluent Emoji We select an additional 256 images at random from Microsoft's Fluent Emoji [21] collection. This dataset features colorful emojis with rich gradient details. We test on this dataset to elaborate on our method's ability to match up with target images using fewer paths compared to previous work [16, 20].

Iconfont A set of 128 vector arts are fetched from the online Iconfont Illustration Library [13]. Images in the dataset contain an average of 1020 paths with great details but no gradients. We use these images to exhibit that our segmentation-guided approach is also suitable for vectorizing complex inputs, and performs on par with or better than previous work even without gradients.

It is worth noticing that our framework is model-free, thus datasets mentioned above are for sole evaluation purposes.

4.3 Qualitative Comparison

With our initial goal of supporting gradient fills in a topology-preserving vectorization approach, we conducted a comparative analysis of the visual quality of our reconstructed vector graphics against LIVE [20]. Our method achieves superior visual quality in vectorization while utilizing the same number of paths as LIVE, as evidenced in Fig. 8.

As depicted in Fig. 9, when presented with input containing gradients, LIVE struggles to accurately vectorize gradient-filled facial features, despite correctly capturing other facial parts. In contrast, our approach accurately reconstructs

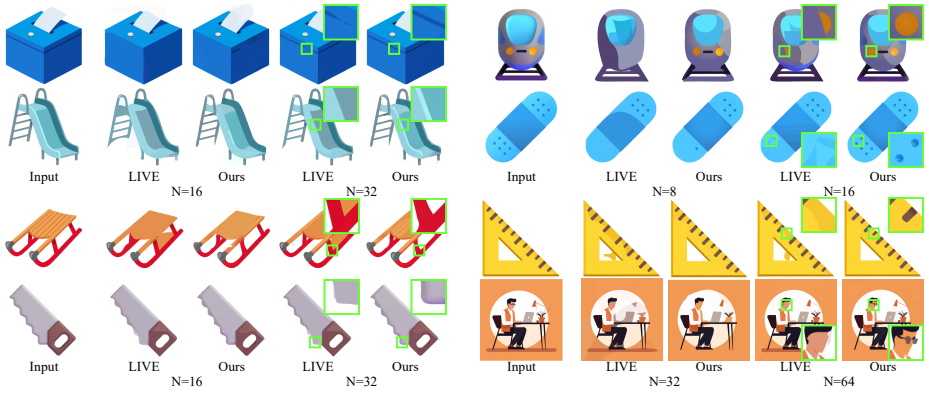


Fig. 8: Qualitative comparisons with LIVE

the input using concise paths. Additionally, attempts to optimize gradient parameters without guidance, as seen in the 'w/o guidance' rows, result in degraded outcomes.

4.4 Quantitative Comparison

We further adopt quantitative metrics in a quantitative comparison with LIVE. We calculated PSNR to measure the difference between the rendered vectorized outputs and their corresponding raster inputs. For Noto and Fluent Emoji datasets, we add $\text{clamp}(2^{i-2}, 1, 32)$ paths at i-th initialization to reach a total of 256 paths. For the Iconfont dataset, 512 paths are used for each image with $\text{clamp}(2^{i-2}, 1, 64)$ paths being added at i-th epoch, for its higher complexity.

As reported in Fig. 10, our method achieves generally faster convergence than LIVE, especially when a small number of paths are added. Our segmentation-guided framework can capture large-scale gradient features where LIVE tends to emit superfluous paths. With excessive paths being added, both methods yield the same level of quality in terms of PSNR.

4.5 Layer Decomposition

The framework we propose captures layer-wise structure during our progressive vectorization. The segmentation-guided initialization prioritizes segments that are more significant in terms of accumulated error, while details with relatively small errors are captured later by our adaptive gradient-aware segmentation, as shown in Fig. 4. Given an input with a clear hierarchy, these progressively added ordered paths resemble the handcrafted vector graphics, creating an easy-to-edit output, as shown in Fig. 11.

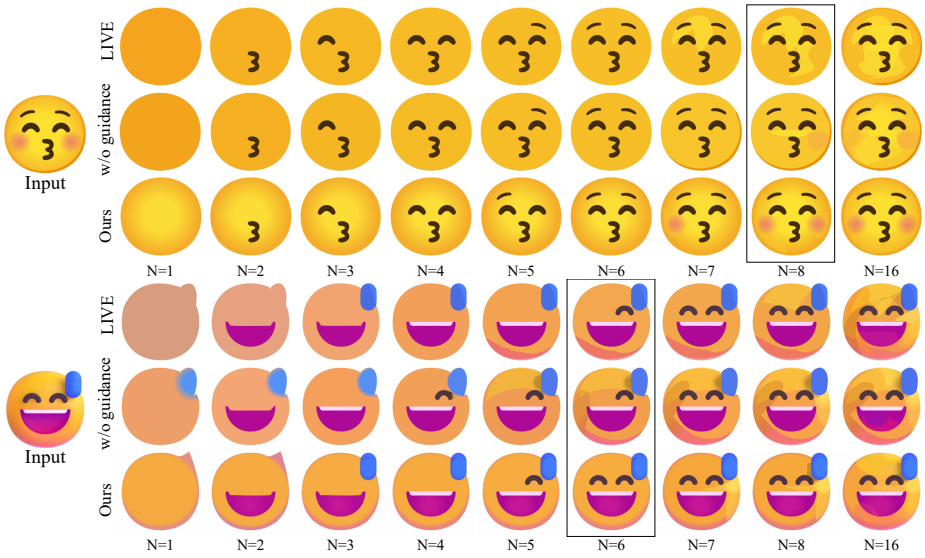


Fig. 9: A comparison on number of paths needed to reconstruct major components. 'w/o guidance' refers to gradient fills being added without our proposed segmentation guidance. For the first input (*U+1F61A* from Noto Emoji), our method vectorizes all elements with 8 paths, while adding more paths does not deteriorate the output. For the second input (*U+1F605* from Fluent Emoji), ours reconstructs the facial parts with 6 paths and achieves a close gradient effect to the input using 16 paths.

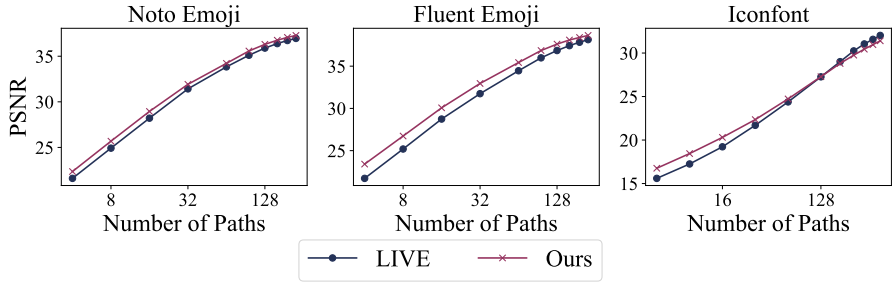
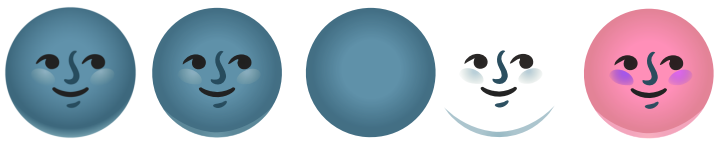


Fig. 10: Quantitative comparison between our method and LIVE. Our method achieves significantly lower error when the numbers of paths are small and converge to equal performance with excessive paths being added.



(a) Source **(b)** Vectorized **(c)** Decomposed layers **(d)** Edited

Fig. 11: Decomposed layers and recoloring

Table 1: Results collected from our user study. The row 'Total' stands for the total number of answers collected for all the images in the dataset vectorized given the number of paths. As shown by the results, users in general prefer the vectorization by our method.

(a) Noto Emoji

#Paths	8	16	32	64	Overall
LIVE	39.0%	39.0%	40.8%	43.0%	40.4%
Ours	**61.0%**	**61.0%**	**59.2%**	**57.0%**	**59.6%**
Total	449	441	441	454	1785

(b) Fluent Emoji

#Paths	8	16	32	64	Overall
LIVE	40.2%	38.4%	30.8%	29.3%	34.7%
Ours	**59.8%**	**61.6%**	**69.2%**	**70.7%**	**65.3%**
Total	433	445	452	426	1756

(c) Iconfont

#Paths	32	64	128	256	Overall
LIVE	32.2%	42.7%	48.5%	44.3%	42.1%
Ours	**67.8%**	**57.3%**	**51.5%**	**55.7%**	**57.9%**
Total	214	241	227	237	919

Table 2: Results of our ablation study in PSNR. 'G' in the header is for 'with gradients' and 'S' is for 'with segmentation guidance'. The first row, with neither gradients nor segmentation, stands for LIVE.

(a) Noto Emoji

G	S	N=8	16	32	64	128	256
		24.92	28.21	31.42	33.83	35.89	37.16
	✓	25.33	28.67	31.63	34.13	36.12	37.34
✓		25.19	28.51	31.80	34.17	36.05	37.29
✓	✓	**25.69**	**28.95**	**31.92**	**34.22**	**36.30**	**37.46**

(b) Fluent Emoji

G	S	N=8	16	32	64	128	256
		25.19	28.74	31.74	34.46	36.85	38.38
	✓	26.26	29.36	32.22	34.67	36.96	38.52
✓		26.15	29.59	32.64	35.28	37.54	**38.91**
✓	✓	**26.73**	**30.07**	**32.95**	**35.43**	**37.61**	38.84

(c) Iconfont

G	S	N=16	32	64	128	256	512
		19.22	21.69	24.38	27.26	30.25	32.36
	✓	20.02	22.02	24.29	26.86	29.43	31.44
✓		19.42	21.96	24.72	**27.60**	**30.67**	**32.72**
✓	✓	**20.31**	**22.35**	**24.73**	27.28	29.76	31.74

4.6 User Study

We conducted a questionnaire survey to obtain subjective feedback from users. Each participant is presented with 20 questions randomly selected from a total of 2,560 questions, that is, 4 numbers of paths multiplied by 640 images from the three datasets. In each question, participants are shown a raster input and two vectorized versions of it, one using LIVE and the other using our method, with the same number of paths. They are asked to choose the one that looks closer to the original image or, at an equivalent level of similarity, looks more appealing. Figure 12 shows examples.

In our study, 223 people engaged, contributing a total of 4,460 votes. The results, presented in Tab. 1, reveal a clear preference for our approach. In the Fluent Emoji dataset, our method is notably preferred for these images with rich gradients. Despite the absence of gradients in many images from Noto Emoji and Iconfont, our method remains the preferred choice.

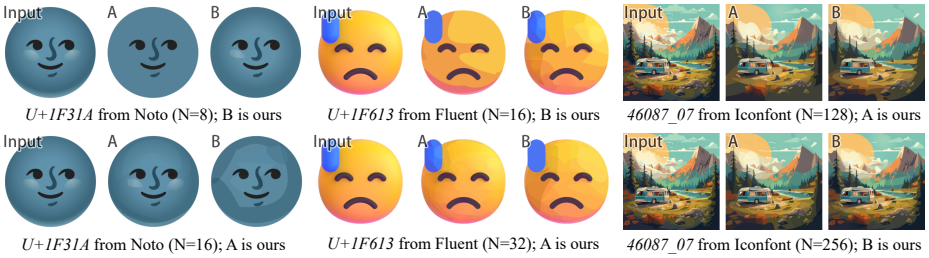


Fig. 12: Examples from our questionnaire. Participants are asked to choose the better one for each pair of outputs.

4.7 Ablation Study

To verify the effectiveness of our contributions, we conducted an ablation study, by vectorizing without gradients, without segmentation guidance, or both, as shown in Fig. 13. The similarity between vectorized results and their corresponding inputs is measured by PSNR. Results are tabulated in Tab. 2.

Gradients For images with rich gradient effects, such as those in Fluent Emoji, the introduction of gradients significantly improves the outcome. For Noto Emoji, where most images do not have gradients, the improvement is less noticeable. Iconfont is an interesting case; although its paths are filled with solid colors, a form of approximate color transition is achieved through multiple paths with progressively changing colors. Introducing gradients also contributes to a noticeable enhancement.

Segmentation guidance Compared to solely introducing gradients, incorporating segmentation guidance has proven more effective. As illustrated in Tab. 2, a combination of gradients and segmentation yields superior results on both Noto and Fluent datasets. For the Iconfont dataset, the imitated color transition is

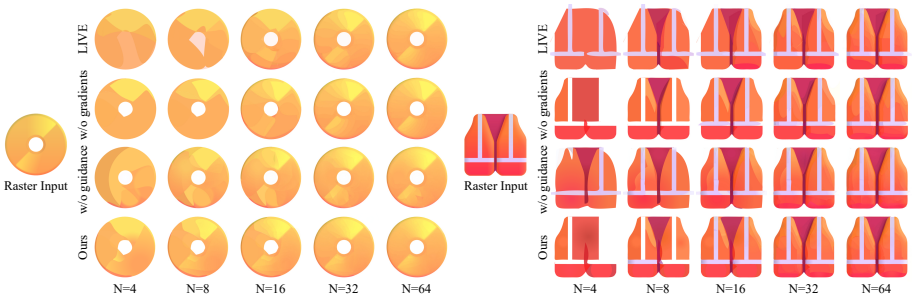


Fig. 13: Comparison between four combinations tested in the ablation study. When neither gradients nor segmentation guidance is applied, our method works the same as LIVE (1st row).

considered a gradient by the segmentation; thus, with fewer paths, the guided variants achieve a more approximate visual effect through large gradient paths. When more paths are added, using smaller, precise solid color paths yields better results, while the guided vectorization remains competitive in terms of accuracy, and is preferred in the user study.

4.8 Limitations and Future Work

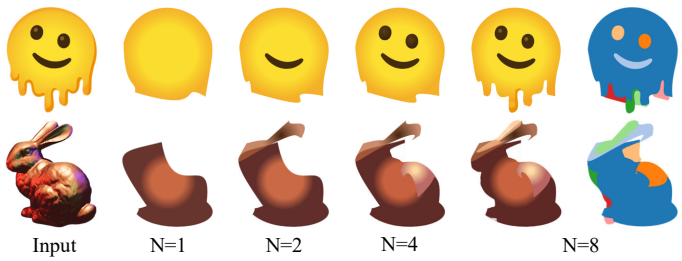


Fig. 14: Intricate contours that our simple initial paths fail to converge to. Extra paths are added to mask the overflowing colors.

While our segmentation guidance effectively captures large-scale features, the initial shape, consisting of a loop formed by four connected Bézier curves, encounters challenges in accurately fitting intricate shapes, as illustrated in Figure 14. Achieving a balance between the number of paths and the number of control points per path needs further deliberation.

Additionally, our implementation employs radial gradients with two color stops for simplicity, since a linear gradient can be interpreted as a radial gradient with its center positioned outside the path. Our method is not confined to a specific type of gradient. A more comprehensive implementation could involve dynamically determining gradient types and adjusting the number of color stops accordingly.

5 Conclusion

In this paper, we propose a segmentation-guided layer-wise vectorization framework, which synthesizes vector images by progressively adding paths and filling the paths with radial gradients. We design a gradient-aware segmentation method out of traditional algorithms to guide our novel initialization approach and newly designed loss function to address the obstacles in optimizing gradient parameters. We test our method on Noto [9] and Fluent Emoji [21] to show its capability in decomposing raster images into concise layers of gradient-filled paths when the topology is clear, and on Iconfont [13] to demonstrate its effectiveness in case of complex inputs. Our method shows superior performance compared to previous work.

Acknowledgements

This work was supported by the NSFC under Grant 62072271.

References

1. Baksteen, S.D., Hettinga, G.J., Echevarria, J., Kosinka, J.: Mesh colours for gradient meshes. STAG: Smart Tools and Applications in Graphics (2021)
2. Beucher, S., Meyer, F.: The morphological approach to segmentation: the watershed transformation. Mathematical morphology in image processing **34**(1993), 49 (1993)
3. Carlier, A., Danelljan, M., Alahi, A., Timofte, R.: Deepsvg: A hierarchical generative network for vector graphics animation. Advances in Neural Information Processing Systems **33**, 16351–16361 (2020)
4. Comaniciu, D., Meer, P.: Mean shift: A robust approach toward feature space analysis. IEEE Transactions on pattern analysis and machine intelligence **24**(5), 603–619 (2002)
5. Du, Z.J., Kang, L.F., Tan, J., Gingold, Y., Xu, K.: Image vectorization and editing via linear gradient layer decomposition. ACM Transactions on Graphics (TOG) **42**(4), 1–13 (2023)
6. Egiazarian, V., Voynov, O., Artemov, A., Volkhonskiy, D., Safin, A., Taktasheva, M., Zorin, D., Burnaev, E.: Deep vectorization of technical drawings. In: European conference on computer vision. pp. 582–598. Springer (2020)
7. Favreau, J.D., Lafarge, F., Bousseau, A.: Photo2clipart: Image abstraction and vectorization using layered linear gradients. ACM Transactions on Graphics (TOG) **36**(6), 1–11 (2017)
8. Frans, K., Soros, L., Witkowski, O.: Clipdraw: Exploring text-to-drawing synthesis through language-image encoders. Advances in Neural Information Processing Systems **35**, 5207–5218 (2022)
9. Noto emoji, `https://github.com/googlefonts/noto-emoji` Accessed: 2023-09-19
10. Ha, D., Eck, D.: A neural representation of sketch drawings. In: International Conference on Learning Representations (2018)
11. Haralick, R.M., Sternberg, S.R., Zhuang, X.: Image analysis using mathematical morphology. IEEE transactions on pattern analysis and machine intelligence (4), 532–550 (1987)
12. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation **9**(8), 1735–1780 (1997)
13. Vector illustrations library, `https://www.iconfont.cn/illustrations/index` Accessed: 2023-11-02
14. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: International Conference on Learning Representations (2015)
15. Kingma, D.P., Welling, M.: Auto-Encoding Variational Bayes. In: International Conference on Learning Representations (2014)
16. Li, T.M., Lukáč, M., Gharbi, M., Ragan-Kelley, J.: Differentiable vector graphics rasterization for editing and learning. ACM Transactions on Graphics (TOG) **39**(6), 1–15 (2020)
17. Liao, Z., Hoppe, H., Forsyth, D., Yu, Y.: A subdivision-based representation for vector image editing. IEEE Transactions on Visualization and Computer Graphics **18**(11), 1858–1867 (2012)

18. Liu, Y.T., Zhang, Z., Guo, Y.C., Fisher, M., Wang, Z., Zhang, S.H.: Dualvector: Unsupervised vector font synthesis with dual-part representation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 14193–14202 (2023)
19. Lopes, R.G., Ha, D., Eck, D., Shlens, J.: A Learned Representation for Scalable Vector Graphics. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 7929–7938. IEEE, Seoul, Korea (South) (Oct 2019)
20. Ma, X., Zhou, Y., Xu, X., Sun, B., Filev, V., Orlov, N., Fu, Y., Shi, H.: Towards layer-wise image vectorization. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 16314–16323 (2022)
21. Fluent emoji, `https://github.com/microsoft/fluentui-emoji` Accessed: 2023-10-27
22. Orzan, A., Bousseau, A., Winnemöller, H., Barla, P., Thollot, J., Salesin, D.: Diffusion curves: a vector representation for smooth-shaded images. ACM Transactions on Graphics (TOG) **27**(3), 1–8 (2008)
23. Otsu, N.: A threshold selection method from gray-level histograms. IEEE transactions on systems, man, and cybernetics **9**(1), 62–66 (1979)
24. Reddy, P., Gharbi, M., Lukac, M., Mitra, N.J.: Im2vec: Synthesizing vector graphics without vector supervision. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 7342–7351 (2021)
25. Reddy, P., Zhang, Z., Wang, Z., Fisher, M., Jin, H., Mitra, N.: A multi-implicit neural representation for fonts. Advances in Neural Information Processing Systems **34**, 12637–12647 (2021)
26. Richardt, C., Lopez-Moreno, J., Bousseau, A., Agrawala, M., Drettakis, G.: Vectorising bitmaps into semi-transparent gradient layers. Computer Graphics Forum (Proceedings of EGSR) **33**(4), 11–19 (July 2014)
27. Rombach, R., Blattmann, A., Lorenz, D., Esser, P., Ommer, B.: High-resolution image synthesis with latent diffusion models. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 10684–10695 (2022)
28. Su, H., Liu, X., Niu, J., Cui, J., Wan, J., Wu, X., Wang, N.: Marvel: Raster gray-level manga vectorization via primitive-wise deep reinforcement learning. IEEE Transactions on Circuits and Systems for Video Technology (2023)
29. Sun, J., Liang, L., Wen, F., Shum, H.Y.: Image vectorization using optimized gradient meshes. ACM Transactions on Graphics (TOG) **26**(3), 11–es (2007)
30. Tian, X., Günther, T.: A survey of smooth vector graphics: Recent advances in representation, creation, rasterization and image vectorization. IEEE Transactions on Visualization and Computer Graphics (2022)
31. Van der Walt, S., Schönberger, J.L., Nunez-Iglesias, J., Boulogne, F., Warner, J.D., Yager, N., Gouillart, E., Yu, T.: scikit-image: image processing in python. PeerJ **2**, e453 (2014)
32. Wang, Y., Lian, Z.: Deepvecfont: Synthesizing high-quality vector fonts via dual-modality learning. ACM Transactions on Graphics (TOG) **40**(6), 1–15 (2021)
33. Xia, T., Liao, B., Yu, Y.: Patch-based image vectorization with automatic curvilinear feature alignment. ACM Transactions on Graphics (TOG) **28**(5), 1–10 (2009)
34. Xie, G., Sun, X., Tong, X., Nowrouzezahrai, D.: Hierarchical diffusion curves for accurate automatic image vectorization. ACM Transactions on Graphics (TOG) **33**(6), 1–11 (2014)
35. Yang, M., Chao, H., Zhang, C., Guo, J., Yuan, L., Sun, J.: Effective clipart image vectorization through direct optimization of bezigons. IEEE transactions on visualization and computer graphics **22**(2), 1063–1075 (2015)

36. Zhu, H., Cao, J., Xiao, Y., Chen, Z., Zhong, Z., Zhang, Y.J.: Tcb-spline-based image vectorization. ACM Transactions on Graphics (TOG) **41**(3), 1–17 (2022)