Illustrator's Depth: Monocular Layer Index Prediction for Image Decomposition

Supplementary Material

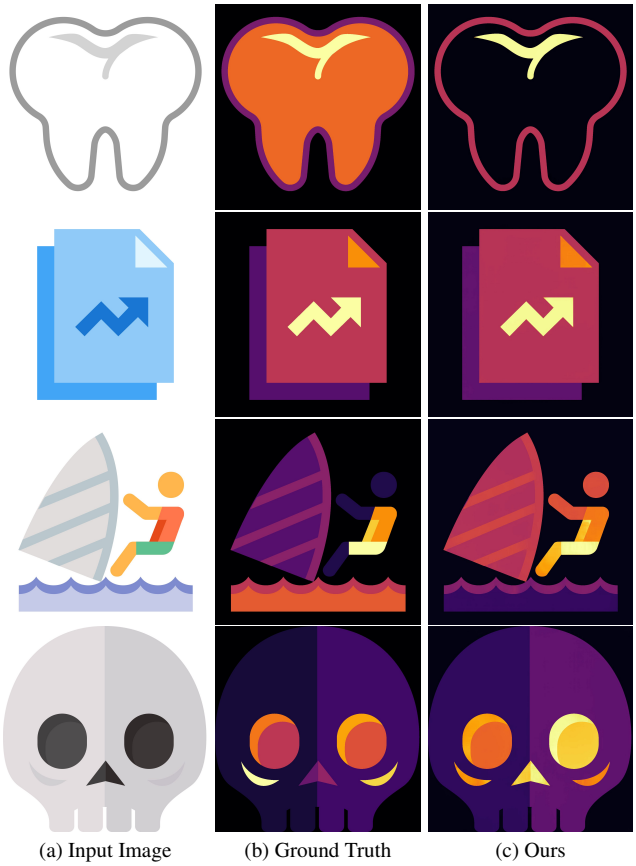
This supplementary material provides additional details, results, and comparisons to complement our CVPR paper on *Illustrator's Depth*. The reader is also encouraged to browse a number of additional files that we provide with our submission: an interactive layer visualization tool (*interactive.html*) to compare various vectorization methods on the two-cherry example from Fig. 3, an *mmsvg-100.html* file containing all the results of our approach and previous methods on the MMSVG-100 dataset, a *showcase.html* file focusing on 31 other complex examples of various natures (paintings, illustrations, natural images) demonstrating the strength of our approach, as well as a video summarizing our approach and its benefits.

7. Evaluation on SVGX

While we trained our model on (a curated subset of) the MMSVG-Illustration dataset [58], we also evaluated our layer index predictions on the SVGX-Core-250k dataset curated by [55] for completeness. Similar to MMSVG, we randomly select 100 images for quantitative analysis. As shown in Tab. 4 and Fig. 11, our model demonstrates strong generalization and maintains excellent performance.

Table 4. **Evaluation of our method on different datasets predicted depth.** Raw outputs of the network.

	Order ↑	MAE ↓	MSE ↓
MMSVG	0.987	0.12	0.26
SVGX-Core-250k	0.984	0.16	0.53



(a) Input Image (b) Ground Truth (c) Ours

Figure 11. **Evaluation of the inferred layer indices.** When evaluated on the SVGX-Core-250k dataset, our method predicts a satisfactory illustrator's depth even if some conventions are different from in our training dataset (for instance, the outline of the tooth is placed *below* the filled-in shape).

8. Ablation Studies

We present additional qualitative results in Fig. 12 to complement the quantitative findings reported in Table 2 of the main paper. While data cleaning and the use of depth priors lead to pronounced improvements, the choice of layer indices vs. disparity space (d vs. $1/d$) yields more subtle effects, yet still provides noticeable gains in these examples.

9. Details on our vectorization pipeline

Our vectorization tests were performed on a pipeline combining VTracer [31] and Potrace [44] with our contributions. Specifically, illustrator's depth based vectorization is achieved as follows:

1. We find color-constant clusters using VTracer (Fig. 13b). The values of several hyper-parameters are important, such as *filter_speckle* to suppress noise, *color_precision* and *layer_difference* to accurately split the image in distinct regions. All of them are provided in our code.
2. Instead of relying on VTracer's heuristics to sort the clusters, we leverage our predicted illustrator's depth (Fig. 13c) for layering, assigning the cluster's depth order to the median of the predicted depth for each cluster.
3. Cluster grouping is important to ensure a well-layered, compact output. After sorting, we further merge layers with neighboring indices if their RGB colors are within a certain threshold $\tau = 0.05$ in the L^2 norm. This results in an ordered clustering image $C \in [1, ...N]^{H \times W}$ (Fig. 13d).
4. While it doesn't affect the final rendering, filling holes and bridging gaps yields simpler, overlapping layers that are compact and easy to edit. Given a cluster with in-

(a) GT (b) W/o Data Cleaning (c) Ours

(d) GT (e) W/ Direct Index (f) Ours

(g) GT (h) W/o Depth Prior (i) Ours

Figure 12. **Ablation studies.** Data cleaning (top row) and direct indexing (middle row) ease the burden of the model, resulting in cleaner predictions, while using a depth prior initialization (bottom row) significantly improves our model's performance.

(a) GT (b) 9 clusters (c) Ours (d) 5 Clusters

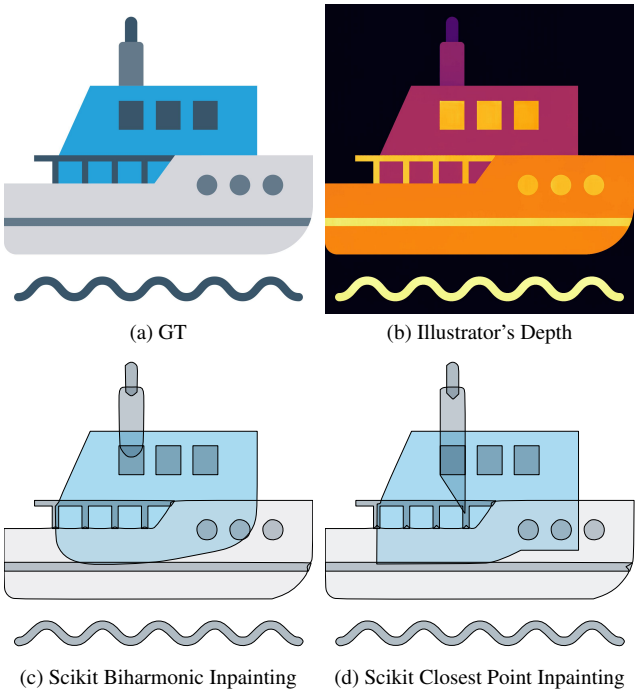
Figure 13. **Depth-aware clustering.** Given an input image (a), VTracer [31] provides a list of color-constant clusters (b). We order these clusters based on our predicted depth map (c) and merge them to form the final decomposition (d).

dex n, we create a binary mask $\mathbf{1}_{C[i,j]>n}$, and inpaint the missing regions of $\mathbf{1}_{C[i,j]==n}$ using off-the-shelf algorithms (see Sec. 10 and Fig. 14).

5. This layer collection is then vectorized with Potrace [44] and assembled to form the final vector graphics.

10. Inpainting

While not part of our contributions, we also show examples of inpainting strategies once our layer index prediction has been generated, see Fig. 14. For vector graphics, we rely on fast, off-the-shelf algorithms provided by Scikit-image [50]. We experimented with two variants in order to fill the missing regions: one that interpolates using the nearest unmasked point, and another based on biharmonic interpolation. Depending on the application, users may prefer one approach over the other: the biharmonic method produces smoother curves, whereas the closest-point interpolation yields sharper, crisper boundaries (see Fig. 14). We generally use the latter in our code due to its faster computational time. While this simple hole-filling approach is sufficient for most vector graphics, data-driven inpainting may be desired for more involved applications, including raster image editing: here again, leveraging off-the-shelf inpainting models (see Fig. 3) offers a solution that doesn't require any additional training.



(a) GT (b) Illustrator's Depth

(c) Scikit Biharmonic Inpainting (d) Scikit Closest Point Inpainting

Figure 14. **Inpainting with Scikit [50].** Using the boat example (a) from Fig. 4, we display two examples of the same layer-wise decomposition produced by our method (b), where inpainting is done via a biharmonic (c) or closest point (d) variant.

11. Prompts for vector-styled images

The main paper shows two text-to-image examples, one using FLUX [19] and one using Nano Banana [10]. For FLUX (Fig. 6 in the main paper), we found that, given a desired object to be drawn (underlined below), a mix of positive and negative prompts provides clean vector-styled raster images that are easy to process with our pipeline; for instance,

```
{ "prompt": "Vector graphics of a simple
    ↪ cheetah head.",
  "prompt_2": "Vector graphics of a simple
      ↪ cheetah head. SVG file. Filled shapes,
      ↪ minimalist design. Abstract.",
```

```
  "negative_prompt": "Gradient, 3D. Small
      ↪ details. Fineline details.",
  "negative_prompt_2": "Gradient, 3D. Small
      ↪ details. Fineline details.",
  "num_inference_steps": 28,
  "num_images_per_prompt": 1
}
```

For Nano Banana (Fig. 7 in the main paper), we simply prompt the model through:

```
{ "prompt": "Vector graphic illustration of a
    ↪ cat. SVG style, blue background. Smooth,
    ↪ flowy shapes."
}
```

12. Comparison with Text2Vector Generators

In addition to the results discussed in Sec. 4.3 of the main paper, more examples of text-to-vector-graphics generations are given in Fig. 15. Both text-to-vector generations using Neural Path Representations [60] and NeuralSVG [29] are based on Score Distillation Sampling (SDS) that relies on a pretrained diffusion model to backpropagate gradients to Bézier curve parameters. Consequently, their generated illustrations are relatively simple and lack fine details (we reproduce the images provided in their articles in Fig. 15). Although LayerTracer [46] employs its own custom diffusion model, it exhibits similar limitations, producing simple emoji-like graphics; note that we used the prompting setup provided in their public repository. In contrast, our method can decompose any output into layered SVG representations, effectively decoupling generation from vectorization — and thus fully leveraging the capabilities of modern generative models. Our modular pipeline, compatible with both Flux [19] and Nano Banana [10], produces detail-rich vector illustrations within seconds (see Sec. 11 for the full prompt configurations).

13. Failure cases

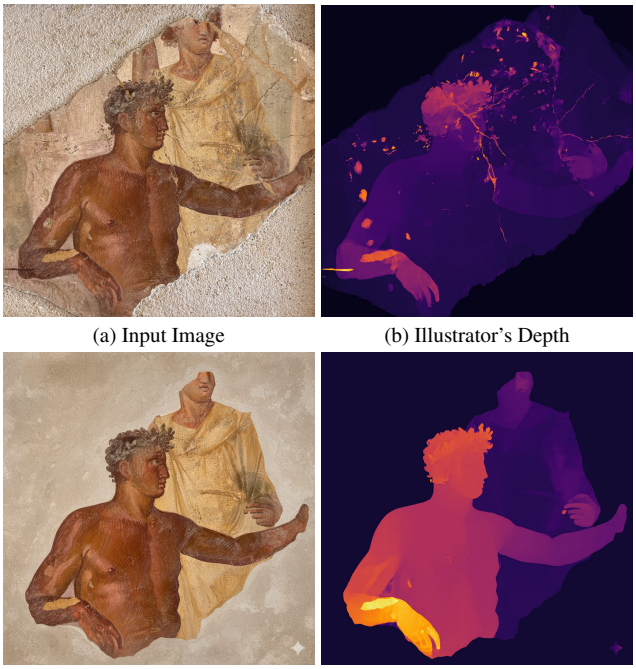
Texture artifacts. Since our model is trained on clean SVG data, a failure case arises when the input image contains canvas textures or defects. These issues can be easily mitigated by using a generative model (e.g., Nano Banana [10]) to clean the image before applying our method (see Fig. 16).

Incorrect ordering. Like any machine learning model, ours can occasionally make mistakes (see Fig. 17, bottom row). Quantitatively, such errors are rare: as shown in Tab. 1, over 98% of randomly sampled pixel pairs are correctly ordered in our experiment with MMSVG.

Foreground Focus. Our training set primarily contains single objects over white backgrounds. Consequently, the model sometimes neglects background elements, which may be undesirable in certain scenarios (see Fig. 17, top row). Future work could address this limitation by training on more complex or synthetic SVG datasets that include background elements.

(a) Ours + [10] (b) LayerTracer [46]
(c) Ours + [19] (d) LayerTracer [46]
(e) Ours + [19] (f) Neural Paths[60]
(g) Ours + [19] (h) NeuralSVG [29]

Figure 15. **Text2Vector models.** Pairing Illustrator's Depth with powerful image generative models produces more complex and detailed illustrations than current text-to-vector diffusion models. In our results (left column), the models are prompted as described in Sec. 11 using "a blue apple", "a cheetah head", "an astronaut riding a horse", and "a colorful peacock".



(a) Input Image (b) Illustrator's Depth

(c) Input Image (cleaned) (d) Illustrator's Depth

Figure 16. **Sensitivity to texture.** The fresco (top left) contains several missing regions and cracks, which our model identifies as foreground elements (top right). If these artifacts are undesired, one can first use Nano Banana [10] to remove defects, then reapply our model to obtain a cleaner result.

14. Depth ordering consistency metric

To compute the depth ordering consistency from a ground-truth illustrator's depth map D and a predicted map D_θ, we adapt the approach of [63] and proceed as follows:

1. we uniformly sample $\frac{H \times W}{50}$ random pairs of pixel locations (i, j) and (k, l) and keep only those corresponding to two different layers in D, i.e., such that $D[i, j] \neq D[k, l]$;
2. we then check whether the relative ordering is preserved by comparing the signs of $(D[i, j] - D[k, l])$ and $(D_\theta[i, j] - D_\theta[k, l])$.
3. Finally, we compute the average consistency score $\bar{s}$ over all pairs by the ratio of preserved ordering over total number of pixel pairs.

This formulation quantifies how effectively the predicted illustrator's depth maintains correct relative depths, independent of absolute scale. This metric, inherently stochastic as it relies on randomly sampled pixel pairs from the image, exhibits strong stability: sampling $50,000$ pairs on 1536×1536 images yielded no significant variations in our experiments (see Fig. 18). And as Tabs. 1-3 from the original paper demonstrate, it offers a complementary measure of layering quality.

(a) Input Image (b) Illustrator's Depth

Figure 17. **Failure cases.** Our models can ignore the background elements, such as the stripes (top row), or incorrectly predict the illustrator's depth: in the bottom row, the leftmost plum should be on top of the leaf rather than behind.

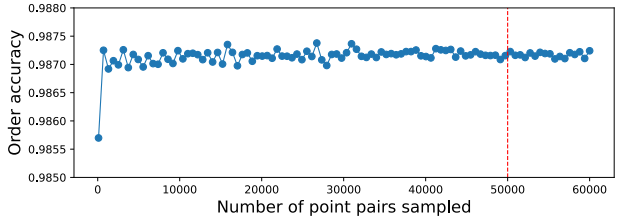


Figure 18. **Stability of order metric.** We plot the order metric when sampling one of our results from 100 to 60K points.

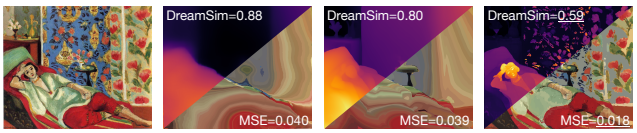
15. Quantitative Evaluation on Natural Images

Quantitative evaluation of Illustrator's Depth in the *absence of ground-truth* (e.g., for bas-relief generation) is challenging. To further quantify how our method generalizes to real paintings, we propose here to (1) compare the predicted depth maps with the input using the human-perception-aligned CLIP Image Score [12] and DreamSim [9], and (2) evaluate piecewise-constant RGB reconstructions after binning into 100 layers with standard metrics. While these tests cannot possibly assess layer index fidelity, they demonstrate that solely training on SVGs improves perceptual scores for artwork images compared to our initialization Depth Pro

(see Tab. 5), which can also be confirmed visually by comparing the different piecewise constant reconstructions on a painting as shown in Fig. 19.

Table 5. **Quantitative Evaluation on 1,000 Random Artworks from WikiArt.**

WikiArt1000	MAE (10^{-2})↓	MSE (10^{-2})↓	SSIM↑	CLIP-I↑	DreamSim↓
Depth Pro [2]	9.716	2.078	0.387	58.7	0.771
Depth A.-v2 [53]	8.238	1.574	0.427	64.1	0.650
Ours	**4.672**	**0.629**	**0.640**	**72.3**	**0.507**



(a) Input (b) Depth Pro (c) Depth A.-v2 (d) Ours

Figure 19. **Evaluation on Paintings.** Compared to traditional Monocular Depth Estimation models, our Illustrator's Depth model better aligns with color-constant regions typical of illustrations.

16. Nearest Neighbors

While our test set is relatively small, nearest neighbor analysis proves that our input images from the test set do not appear in the training set, as demonstrated in Fig. 20.

(a) Test Image (b) NN (RGB) (c) NN (Depth) (d) Test Image (e) NN (RGB) (f) NN (Depth)

Figure 20. **Nearest Neighbor.** For two test images (a) and (d), we show their nearest neighbors in the training set based on pixel-wise RGB and Illustrator's Depth. This confirms their novelty for the network.

17. Additional results

For completeness, we also provide a histogram of the number of layers present in our curated training dataset in Fig. 21, as well as a figure demonstrating another potential use of our illustrator's depth in Fig. 22, where a painting is automatically turned into a multi-layered pop-up card.

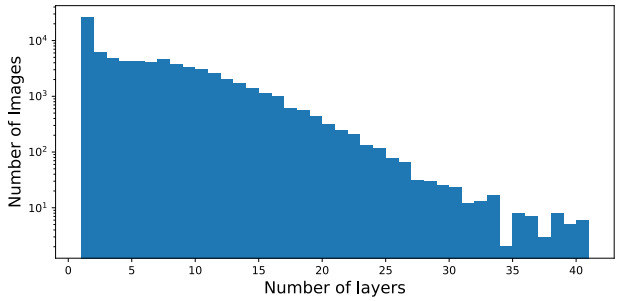


Figure 21. **Number of layers in MMSVG training set.** We plot the histogram of the number of layers in our training dataset. Note that each layer may have many connected components, resulting in a large number of paths.

Figure 22. **Automatic pop-up card generation.** From an image (top left) and our predicted illustrator's depth (bottom left), a multi-layered pop-up card can easily be created using our method — see video for animation.