

## 440 A Ablation Study

441 **The effectiveness of adaptive pruning and densification and layer-wise vectorization strategy.**  
 442 We conduct an ablation study to evaluate the effectiveness of our adaptive pruning and densification  
 443 strategy. Quantitative results are shown in Table 3. To further investigate the flexibility of our  
 444 framework, we incorporate a layer-wise curve addition strategy inspired by LIVE [17], where curves  
 445 are progressively added during training. Although this strategy achieves slightly better reconstruction  
 446 metrics, it doubles the training time compared to our adaptive pruning and densification approach,  
 447 making it less favorable for time-sensitive applications. Fig. 7 shows our result with layer-wise image  
 448 vectorization strategy.

Table 3: An ablation study on adaptive pruning and densification and layer-wise training strategy, evaluated on 512 closed curves with DIV2K dataset [23].

Method	$SSIM^\dagger$	$PSNR^\dagger$	$LPIPS^\downarrow$	$Opt^\downarrow$
No Strategy	0.590	21.10	0.530	<b>8.3 min</b>
– w/ Prune&Densify	0.607	22.11	0.528	8.3 min
– w/ Layer-wise	<b>0.613</b>	<b>22.21</b>	<b>0.521</b>	16.2 min

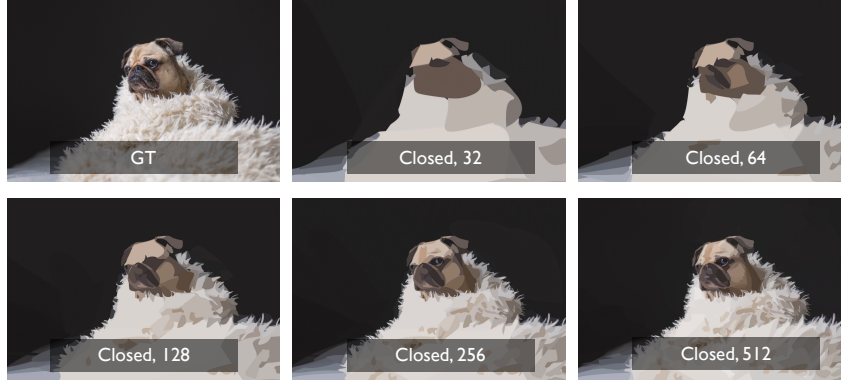


Figure 7: Our Bézier Splatting is fully compatible with the layer-wise vectorization strategies [17].

449 **Comparing different numbers of curves.** We progressively increase the number of curves for  
 450 vectorizing a watercolor image (Fig. 8) with numerous small spots. As the number of curves grows,  
 451 our approach first reconstructs the foreground object, the bird, then gradually refines the smaller spots.  
 452 This demonstrates that our method prioritizes the primary structure before optimizing finer details,  
 453 effectively distributing curves to balance global structure and local texture representation, thanks to  
 454 our adaptive pruning and densification strategy.

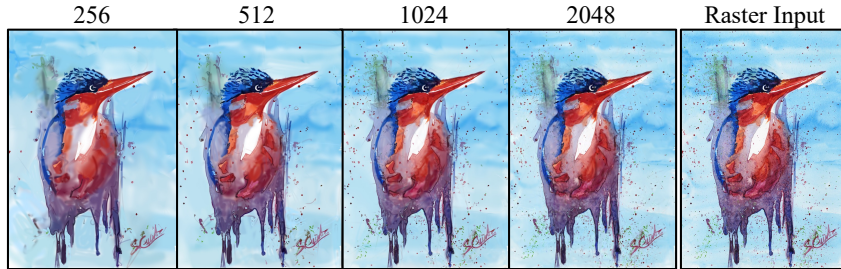


Figure 8: A comparison of different curve numbers by Bézier Splatting.

455 **A systematic evaluation of computation time.** To systematically evaluate the computational  
 456 efficiency of our framework, we report detailed timing statistics across varying configurations of  
 457 interpolation curve numbers and curve resolutions, as summarized in Table 4. The total forward time  
 458 includes two major parts: the Gaussian sampling and the Gaussian splatting. Increasing the number  
 459 of interpolated curves within the closed Bézier curves leads to a near-linear growth in both forward  
 460 and backward runtimes, demonstrating good computational scalability. When fixing the interpolation

number (e.g., at 40) and varying the number of Bézier curves, the forward time increases moderately, reflecting the additional cost of handling finer spatial detail. The splatting stage dominates the forward time at higher number of curves due to the increased number of sampled points per region, while the Gaussian sampling time is the primary bottleneck at lower number of curves. This is attributed to overheads from sequential memory allocation and data transfer operations, which are not effectively parallelized on the GPU, suggesting potential for further optimization.

Table 4: A systematic evaluation of computation time. We test the forward and backward time (ms) under varying numbers of sampled curves within one closed Bézier curve (Inter. #), and the total number of Bézier curves (Curve #).

Inter. #	Curve #	Forward w/ gradient	Sample Gaussians	Splatting	Backward	Render (FPS)
20	2048	10.66	4.12	6.11	21.64	103.45
40	2048	14.79	5.11	9.32	25.06	68.30
80	2048	27.01	8.22	17.74	33.40	37.90
40	256	7.20	3.81	3.09	16.72	150.30
40	512	7.70	3.88	3.67	17.42	144.30
40	1024	10.39	4.06	5.96	19.96	114.30
40	2048	14.79	5.11	9.32	25.06	68.30

## B Convert Bézier Splatting to standard SVG Format

As described in Algorithm 1, we convert our optimized Bézier Splatting representation into a standard SVG file for compatibility with downstream vector editing tools. Each curve is represented by a set of  $3k + 1$  control points  $C_i \in \mathbb{R}^{(3k+1) \times 2}$ , corresponding to  $k$  continuous cubic Bézier segments. These control points are normalized to the range  $[-1, 1]$  and are first transformed into pixel coordinates according to the canvas size  $(W, H)$ .

For each curve  $i$ , we initialize the SVG path string  $d$  with a  $[M \ P_i^0]$  command. We then iteratively construct each cubic segment using three internal control points:  $(p_1, p_2, p_3) = (P_i^{3j+1}, P_i^{3j+2}, P_i^{3j+3})$ , and append the segment as a path command  $[C \ p_1, \ p_2, \ p_3]$  to  $d$ . After all segments are processed, a  $[Z]$  is appended to close the path.

Each path  $d$  is then associated with a fill color  $(r, g, b)$ , computed via sigmoid activation on the optimized feature vector  $\mathcal{F}_i$ , and opacity 1.0. The result is stored in a global path set  $S$ , and all paths are assembled into the final SVG file  $S$ . This output is directly compatible with standard vector graphics tools such as Adobe Illustrator.

---

### Algorithm 1 Convert Bézier Splatting to Standard SVG.

---

**Input:** Optimized control points  $\mathcal{C} \in \mathbb{R}^{N \times (3k+1) \times 2}$ , feature colors  $\mathcal{F} \in \mathbb{R}^{N \times 3}$ , canvas size  $(W, H)$

**Output:** SVG file  $S$  containing cubic Bézier paths

**for** each curve  $i = 1$  to  $N$  **do**

$P_i \leftarrow \frac{\mathcal{C}_i + 1}{2} \cdot (W, H)$

$(r, g, b) \leftarrow \text{sigmoid}(\mathcal{F}_i) \times 255$

Initialize path  $d \leftarrow [M \ P_i^0]$

**for** each segment  $j = 1$  to  $k$  **do**

$(p_1, p_2, p_3) \leftarrow (P_i^{3j+1}, P_i^{3j+2}, P_i^{3j+3})$

$d \oplus [C \ p_1, \ p_2, \ p_3]$

**end for**

$d \oplus Z$

$S \leftarrow S \cup \{\text{path}(d, \text{fill} = (r, g, b), \text{opacity} = 1.0)\}$

**end for**

**Return:** SVG file assembled from all paths

---

## C Bézier Splatting Supports Flexible Attributes of Curves

All 2D Gaussians in our method are generated via a differentiable sampling algorithm, which makes it straightforward to incorporate user-defined shape attributes, such as linear-gradient fills in color or opacity. To demonstrate the extensibility of our model, we evaluate the multi-opacity scheme for open curves: each Bézier segment within an open curve is assigned an independent opacity value. This design enables users to either maintain consistent sampling point appearance across segments or apply customized interpolation strategies. This strategy significantly improves vectorization quality for open curves, as shown in Table 5. Since closed curves are the primary representation format in SVG, we adopt a uniform setting for opacity and color in those regions to ensure a full compatibility with existing vector graphic tools. Nonetheless, users can still follow the above multi-opacity scheme to implement linear-gradient fills for both opacity and color, enhancing the flexibility and expressiveness of vector graphics.

Table 5: An ablation study on the number of opacity parameters per curve, evaluated on 512 open curves with DIV2K dataset [23].

Method	$SSIM^\uparrow$	$PSNR^\uparrow$	$LPIPS^\downarrow$
3-opacity	<b>0.65</b>	<b>23.79</b>	<b>0.50</b>
1-opacity	0.63	22.97	0.51

## D More Comparisons

We present more qualitative comparisons on the DIV2K [23] dataset. As shown in Fig. 9, Fig. 10, Fig. 11, Fig. 12, and Fig. 13, our method shows consistent improvements over the baselines, including better preservation of fine details, higher rendering fidelity, fewer visual artifacts, and more accurate geometric structures. Furthermore, as the number of curves increases, our method demonstrates a significantly improved representational ability, enabling more precise reconstruction of complex shapes, and fine-grained structures.

We further evaluate our method on another natural image dataset, Kodak [13]. Due to the slow vectorization speed of LIVE [17], we compare with DiffVG on open curves only. As shown in Table 6, our method consistently outperforms DiffVG across quantitative metrics including PSNR, SSIM, and LPIPS.

Table 6: A quantitative evaluation on the Kodak dataset [13] with 256 to 1024 curves. We report results on open curves.

	Method	256			512			1024		
		$SSIM^\uparrow$	$PSNR^\uparrow$	$LPIPS^\downarrow$	$SSIM^\uparrow$	$PSNR^\uparrow$	$LPIPS^\downarrow$	$SSIM^\uparrow$	$PSNR^\uparrow$	$LPIPS^\downarrow$
Open	DiffVG	0.601	23.43	0.535	0.645	24.70	0.495	0.699	26.04	0.439
	Ours	<b>0.679</b>	<b>26.18</b>	<b>0.457</b>	<b>0.743</b>	<b>27.90</b>	<b>0.383</b>	<b>0.797</b>	<b>29.24</b>	<b>0.310</b>

## E More Image Vectorization Results

Fig. 14, Fig. 15, and Fig. 16 show more results on natural images from DIV2K [23] and Kodak [13], as well as animation images from DanbooRegion [31], respectively. The results demonstrate that both the global structure and local texture of the images are well reconstructed, highlighting the effectiveness of our approach in capturing fine details and complex shapes.

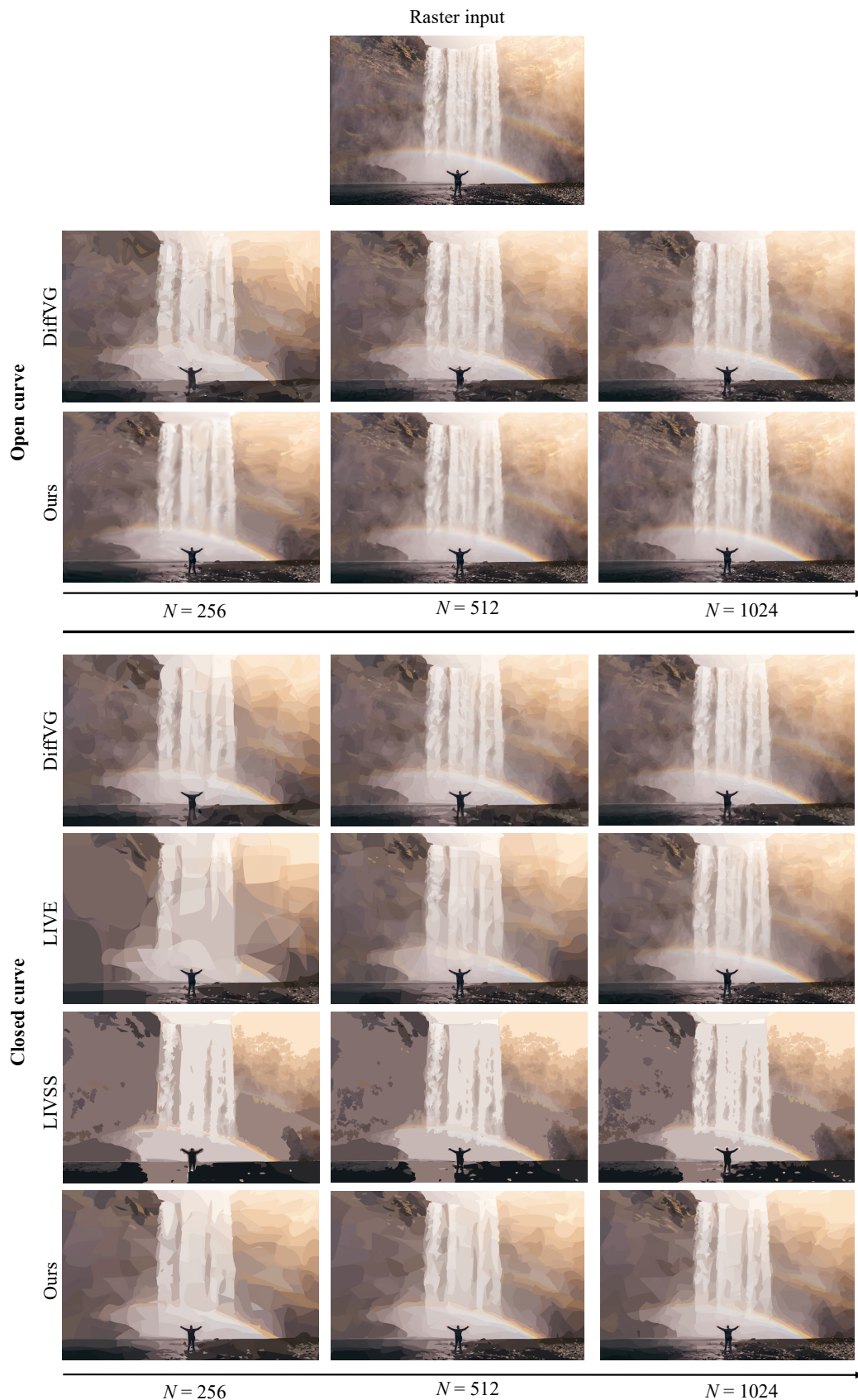


Figure 9: A qualitative comparison of our method and the existing differentiable VG rasterization method on DIV2K dataset [23].



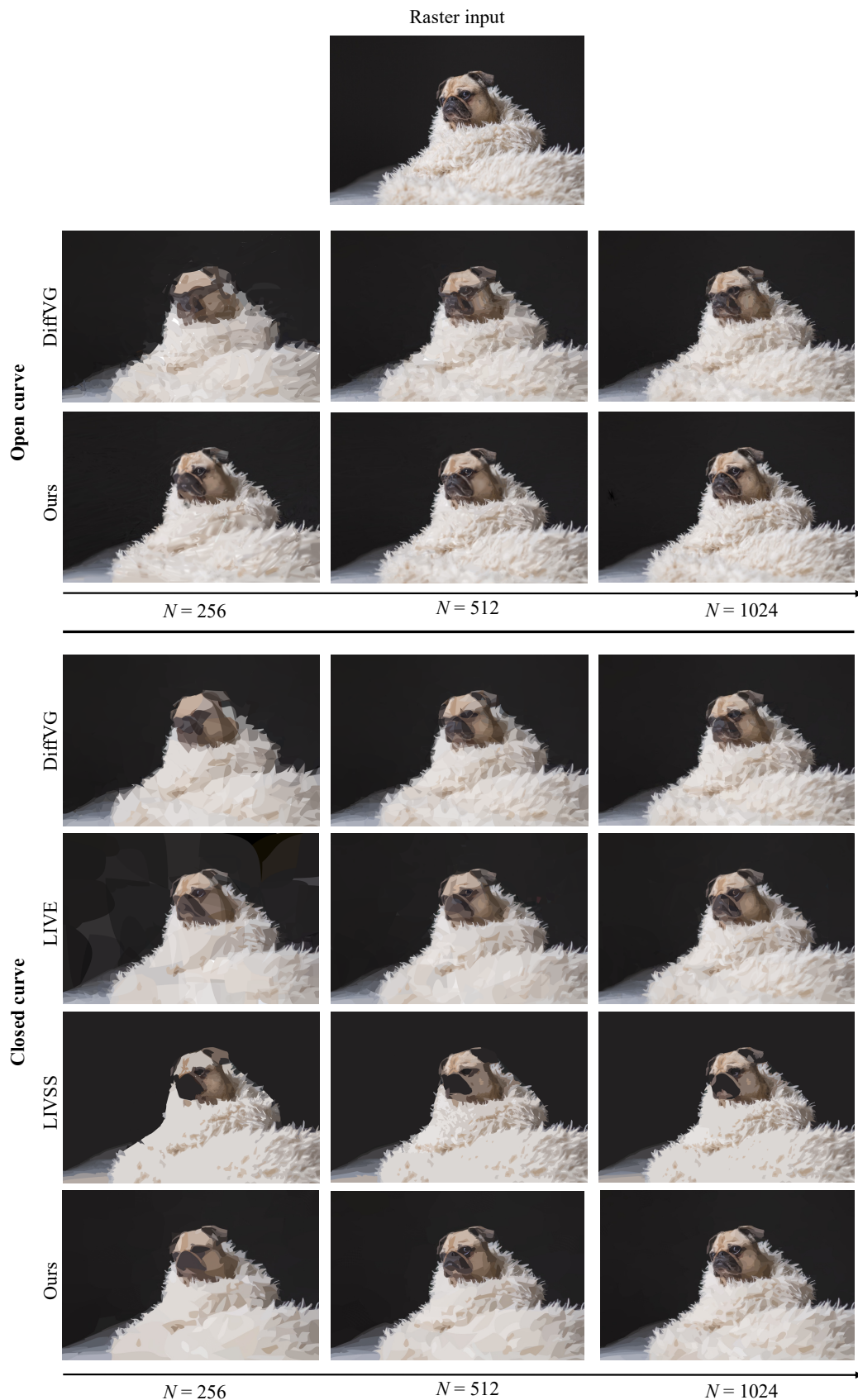


Figure 10: More qualitative comparisons of our method and the existing differentiable VG rasterization method on DIV2K dataset [23].

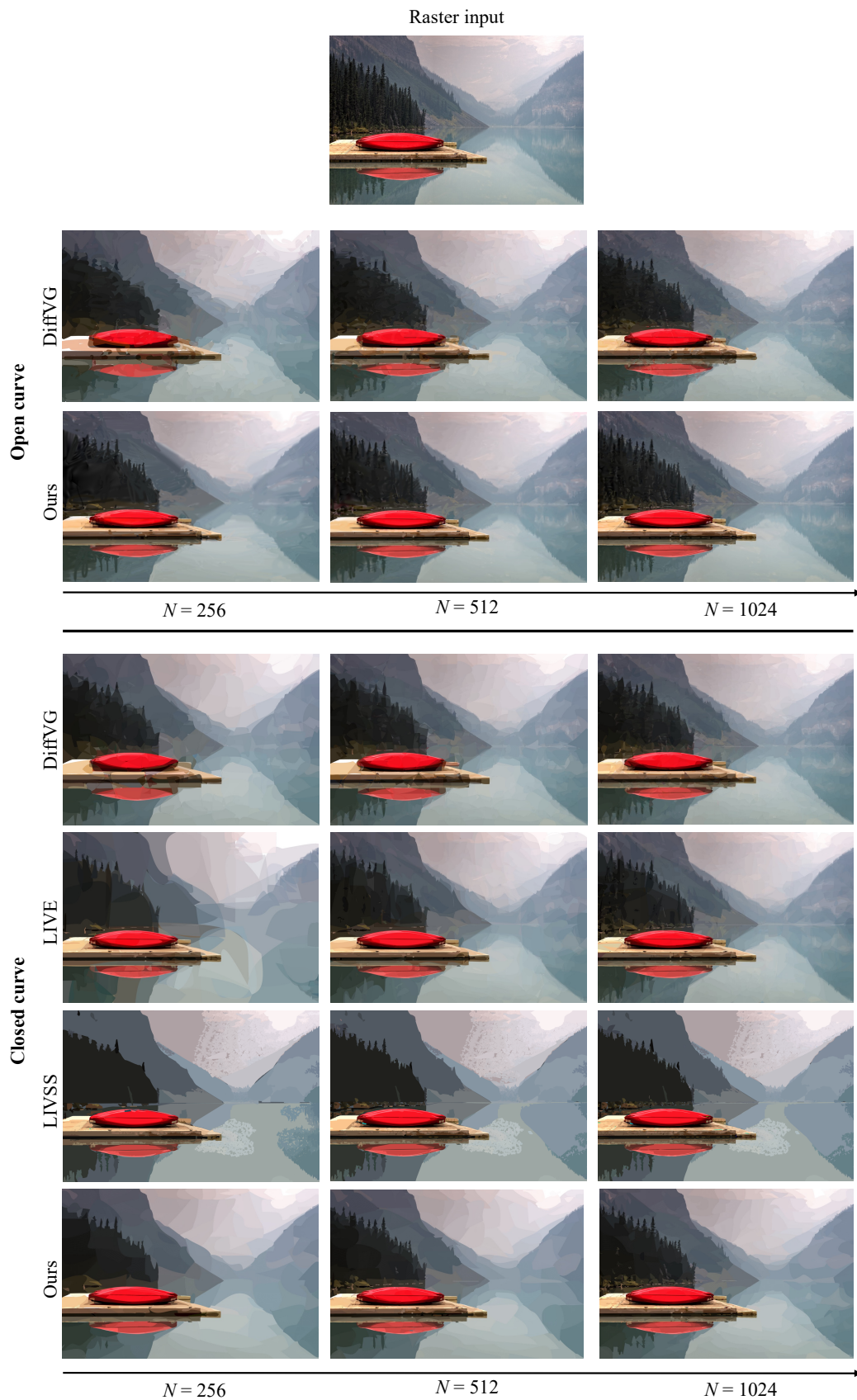


Figure 11: More qualitative comparisons of our method and the existing differentiable VG rasterization method on DIV2K dataset [23].

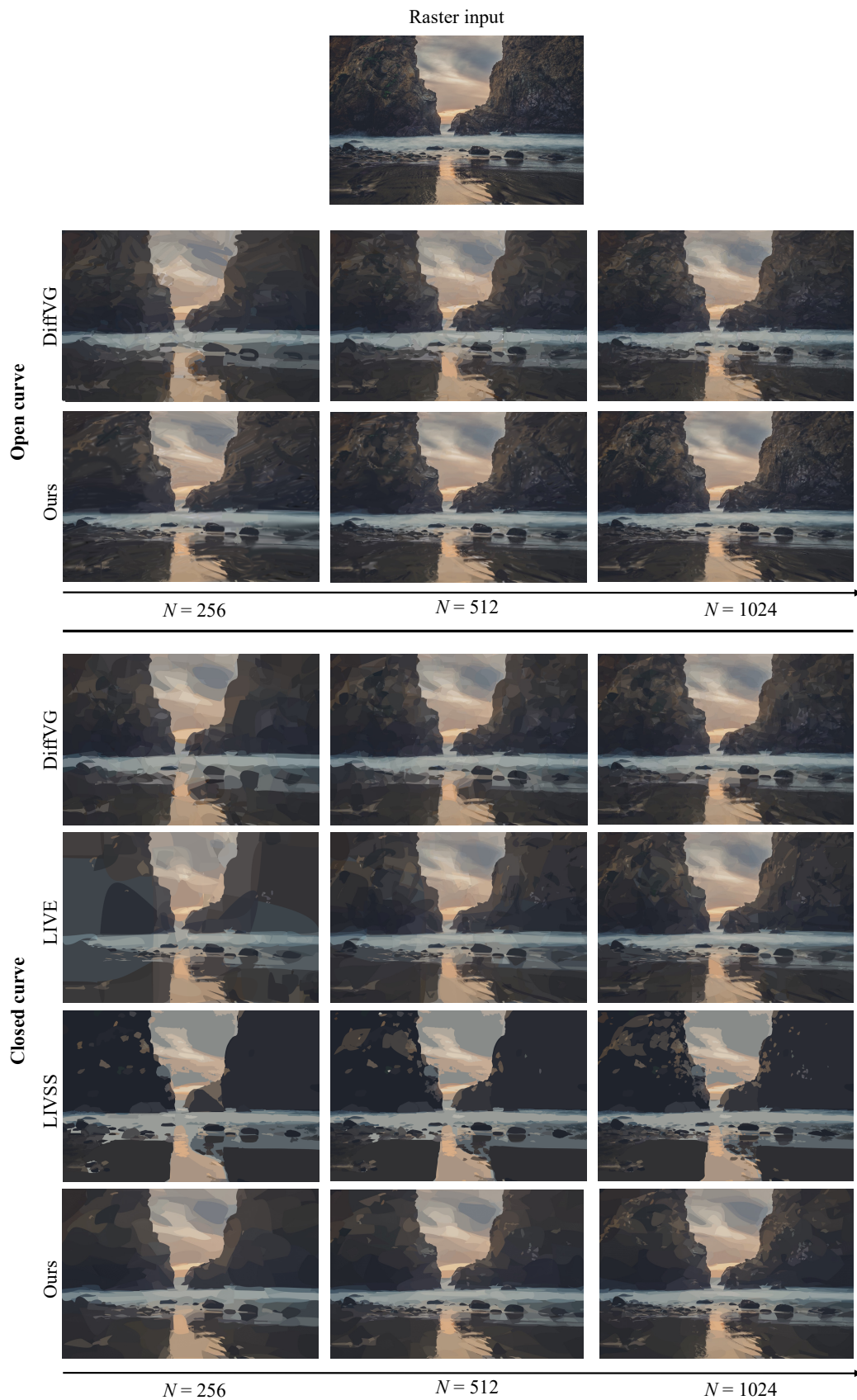


Figure 12: More qualitative comparisons of our method and the existing differentiable VG rasterization method on DIV2K dataset [23].



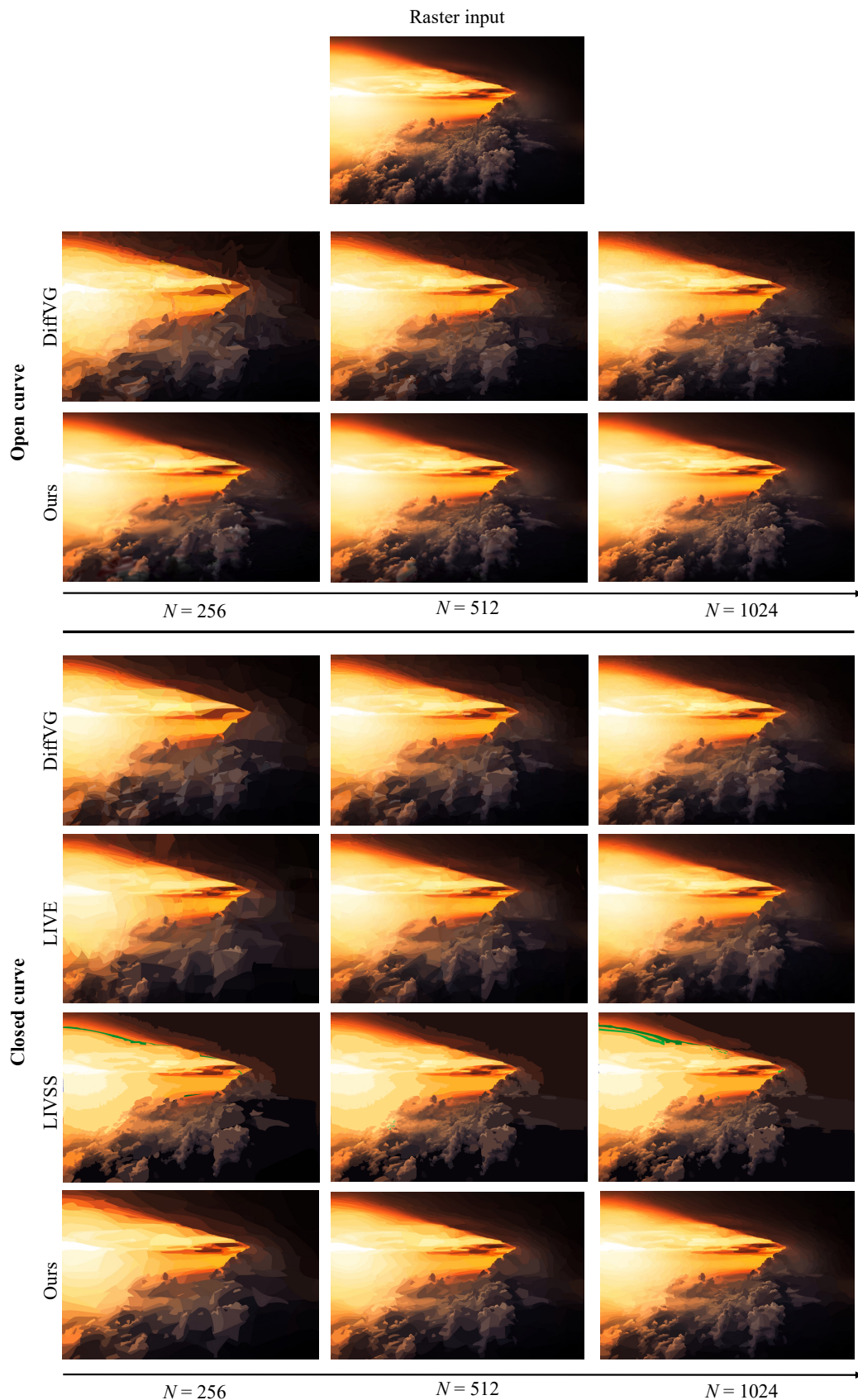


Figure 13: More qualitative comparisons of our method and the existing differentiable VG rasterization method on DIV2K dataset [23].



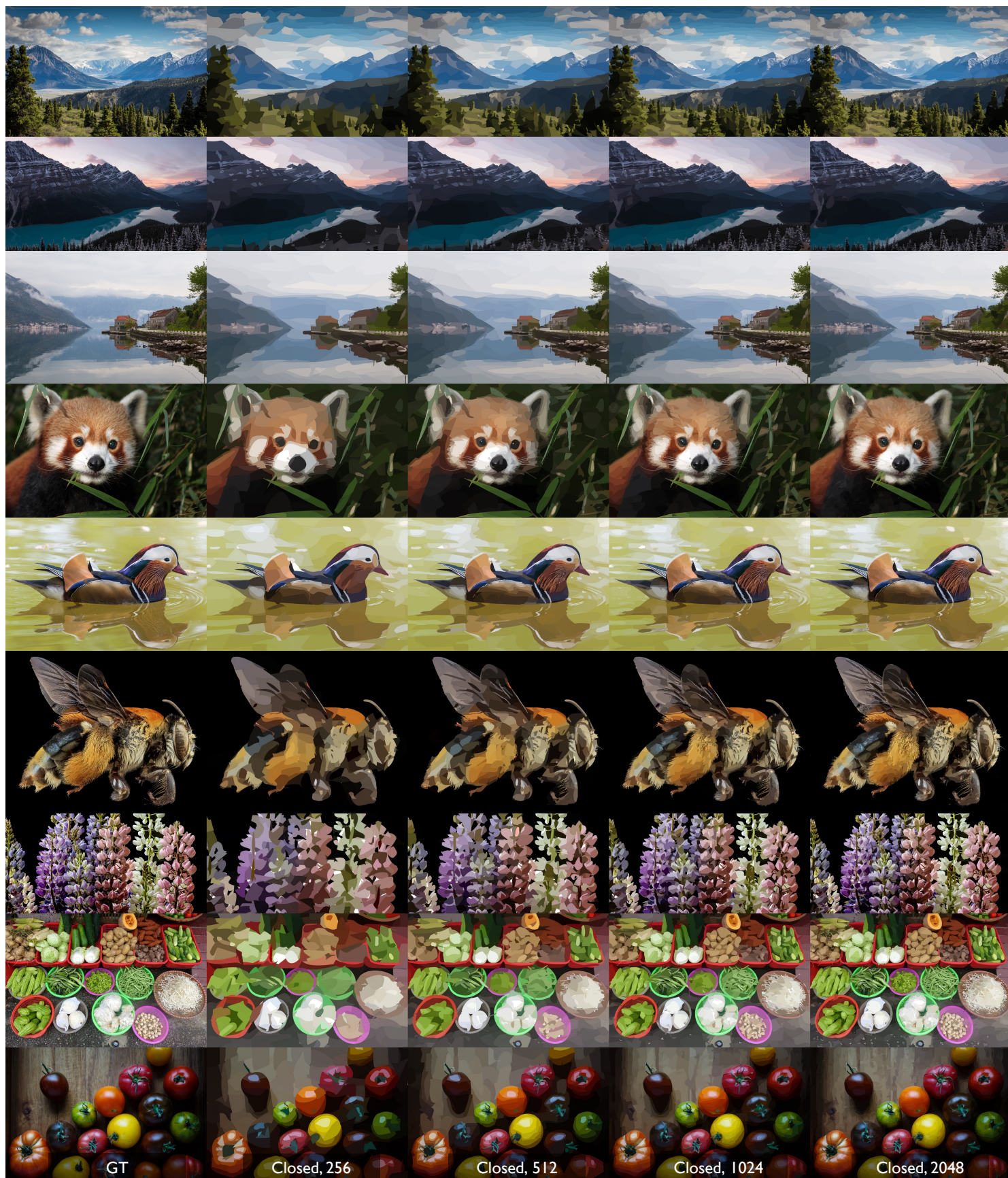


Figure 14: More image vectorization results by B zier Splatting on DIV2K dataset [23].



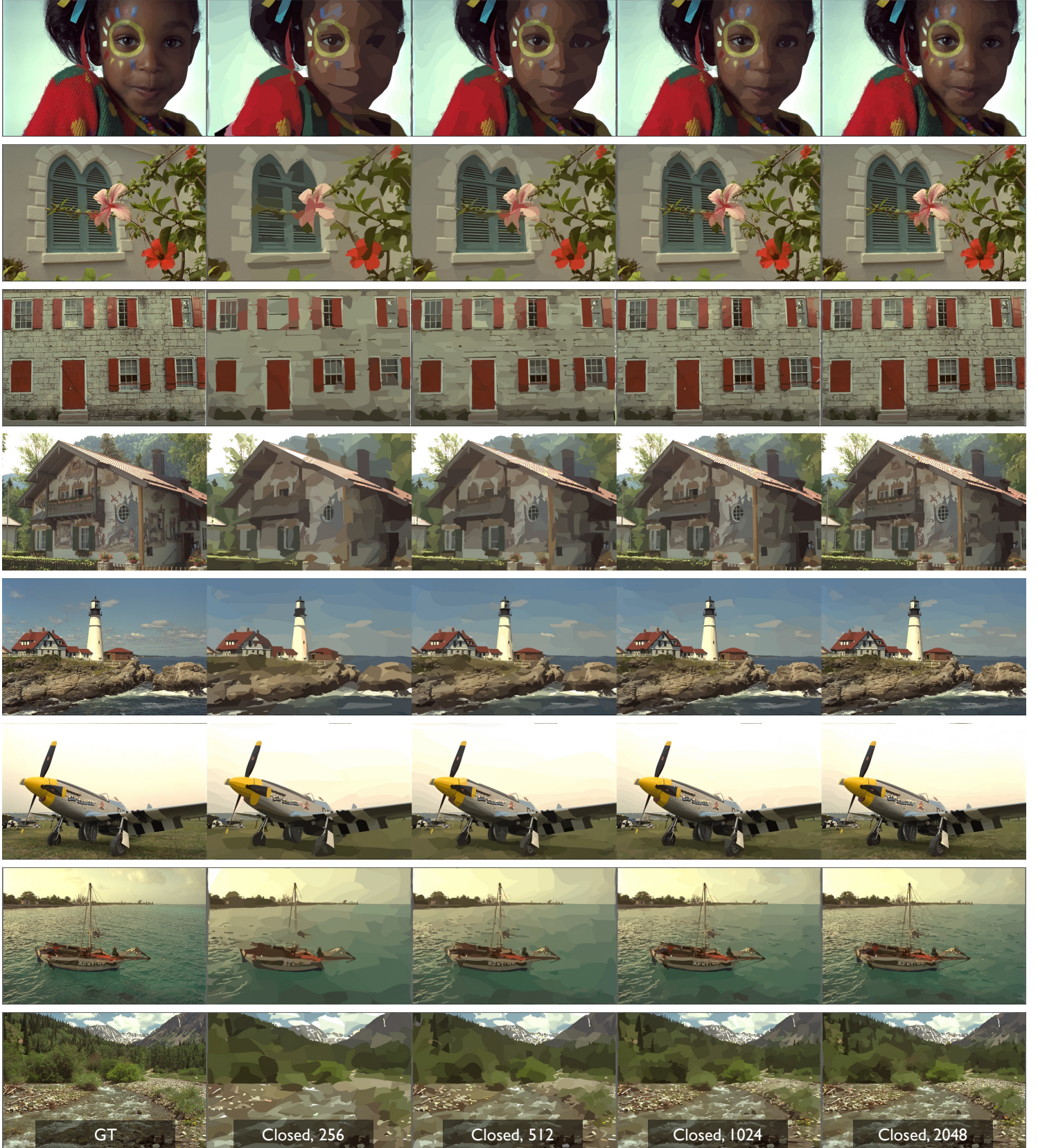


Figure 15: More image vectorization results by Bézier Splatting on Kodak [13] dataset.





Figure 16: More image vectorization results by Bézier Splatting on DanbooRegion dataset [31].