Supplementary Material

Vector Scaffolding: Hierarchical Optimization for Fast and Stable Differentiable Image Vectorization

Jaerin Lee, Kanggeon Lee, Kyoung Mu Lee

A Algorithmic Implementation Details

In the main paper, we introduced *Progressive Stratification* and *Rapid Inflation Scheduling* to orchestrate the multi-scale curve mixture. To ensure full reproducibility, we detail the exact optimization loop in Algorithm 1.

The algorithm highlights our seamless integration of automated pruning and residual-guided densification. At every inflation step ($t \in \{100, 200, \dots\}$), we prune visually negligible curves ($\sigma(\alpha) \leq 0.01$). To maintain the scheduled population growth, we compensate for these pruned curves by spawning an equivalent number of new curves, alongside the exponential inflation governed by r_{num}. Newly spawned curves are strictly restricted by the monotonically decaying radius bound ($R_k = R_{k-1} \times r_{\text{rad}}$) and are initialized on regions with high reconstruction error ($\mathcal{E}$) to encourage spatial separation between generations.

B More Ablation Studies

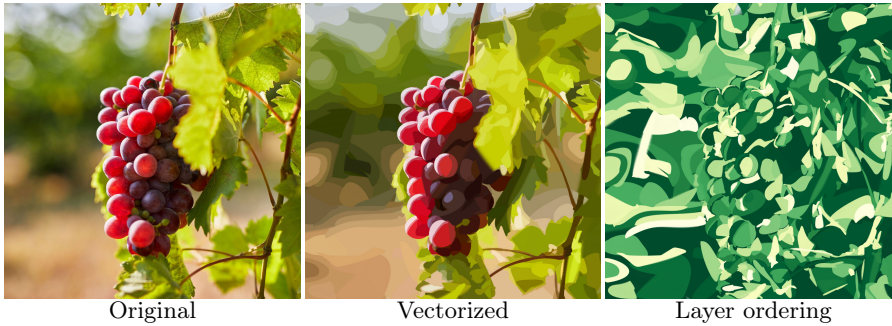

Fig. 7: Layered Primitive Visualization. The deterministic temporal z-ordering induced by *Progressive Stratification* naturally aligns the optimization-induced layer index with the underlying scale hierarchy, so newer fine-scale curves sit on top of coarser base curves without dynamic re-sorting.

Algorithm 1 Vector Scaffolding Optimization Loop

Require: Target Image $\mathcal{I}_{gt}$, Total inflation steps K, Step interval $\Delta T = 100$, Population ratio r_{num}, Scale decay $r_{\text{rad}} = 0.5$.

Initialize base curves $\mathcal{B}$ with base radius R_{base}
$R_{curr} \leftarrow R_{base}$
for $t = 1$ **to** MaxIterations **do**
    Render $\mathcal{I}_{pred} \leftarrow g(\mathcal{B})$
    Compute Loss $\mathcal{L}$ and backpropagate
    if $t \in \{\Delta T, 2\Delta T, \ldots, K\Delta T\}$ **then**
        ▷ *1. Opacity-based Pruning*
        $\mathcal{M}_{prune} \leftarrow \{\mathcal{B}_i \mid \sigma(\alpha_i) \leq 0.01\}$
        Remove curves in $\mathcal{M}_{prune}$ from $\mathcal{B}$
        $N_{removed} \leftarrow |\mathcal{M}_{prune}|$
        ▷ *2. Progressive Stratification*
        $R_{curr} \leftarrow R_{curr} \times r_{\text{rad}}$
        ▷ *3. Residual Error Calculation*
        $\mathcal{E} \leftarrow \|\mathcal{I}_{gt} - \mathcal{I}_{pred}\|_2^2 * \text{GaussianBlur}$
        ▷ *4. Rapid Inflation Scheduling*
        $N_{target} \leftarrow \text{len}(\mathcal{B}) \times r_{\text{num}}$
        $N_{spawn} \leftarrow N_{target} + N_{removed}$
        Sample N_{spawn} coordinates from $\mathcal{E}$
        Spawn new curves at sampled coords with R_{curr}
        Append new curves to $\mathcal{B}$ with Z-ordering
    end if
    Optimizer Step and Zero Grad
end for

Layered Primitive Structure. Figure 7 visualizes the curves of a vectorized image, colored by their generation index. Because the temporal z-order is fixed by construction in *Progressive Stratification*, the resulting primitive stack carries an interpretable LoD-aligned structure rather than the chaotic bounding-box-sorted ordering used by previous splatting-based methods [14].

Editability. To complement the quantitative metrics, we provide a qualitative demonstration of vector-level editability in Figure 8. Because our framework produces a LoD-aligned hierarchy of closed Bézier curves (Fig. 7), users can directly select and modify primitives at a chosen scale. Edits remain localized to the intended region, supporting our claim that hierarchical scaffolding yields improved local editability over flat polygon-soup outputs.

Hyperparameter Sensitivity. We perform a one-at-a-time sweep around the paper defaults to verify that our optimization is robust to hyperparameter choices. Table 3 reports the PSNR sensitivity to (a) the global learning-rate multiplier λ, (b) the radius decay ratio r_{rad}, and (c) the inflation gap ΔT on a 4-image Kodak subset. Across all three knobs, the PSNR at 5 k iterations varies by at most ~ 2.6 dB, with the paper default consistently lying at or near the

Fig. 8: Editability Demonstration. Output of our framework imported into a vector-editing demo built upon our pipeline. The hierarchical scaffold yields path primitives organized by level-of-detail, enabling straightforward selection and local edits at the vector level. We claim improved *local* editability rather than a full semantic-editability solution.

Table 3: Hyperparameter sensitivity analysis on a 4-image Kodak subset (mean PSNR, dB). Bold*: paper default; Δ: max−min range. Performance is stable across a wide range of values around the defaults.

(a) LR multiplier λ						
value	×1	×5	×10	**×50***	×100	Δ
@1 k	23.48	25.79	26.16	**26.95**	26.91	3.47
@5 k	25.83	27.07	27.38	**27.95**	27.84	2.12
(b) Radius decay r_{rad}						
value	0.1	0.25	**0.5***	0.75	1.0 (off)	Δ
@1 k	26.40	26.68	**26.96**	26.93	24.33	2.63
@5 k	27.80	27.92	**27.94**	27.85	25.30	2.64
(c) Inflation gap ΔT						
value	25	50	**100***	200	400	Δ
@1 k	27.27	27.16	**26.97**	23.81	21.39	5.87
@5 k	27.85	27.93	**27.93**	27.97	27.95	0.19

optimum. The largest improvement margin (LR $\Delta = 3.47$ dB at 1 k iterations) confirms that the ×50 learning rate is the most impactful design decision, while performance remains stable across a wide range of r_{rad} and ΔT.

C Visualization of Optimization Videos

To better demonstrate the topological advantages of our proposed method, we capture intermediate renders during the training process. We compare the learning dynamics between the previous state-of-the-art, Bézier Splatting [14], and our Vector Scaffolding side by side across various challenging scenes, with synchronized iteration counts for direct comparison. Note that our method has a slightly more efficient ($\sim$ 25%) iteration loop than the baseline; therefore, running our full algorithm for 5 k iterations is more than 2.5× faster than running

Fig. 9: Optimization trajectory on Kodak `kodim01`. Top: ours; bottom: Bézier Splatting. Columns are matched iterations (∼100, 600, 1600, 4000, 9980). Our method anchors smooth roof/wall regions early, whereas the baseline scatters narrow strokes that never coalesce.

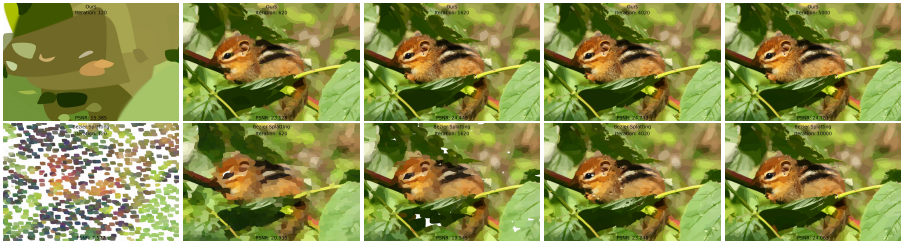

Fig. 10: Optimization trajectory on DIV2K `0294`. Top: ours; bottom: Bézier Splatting. Background foliage and fur texture form coherently in our run, while the baseline keeps redistributing strokes near the subject without locking the surrounding context.

the baseline algorithm for 10 k iterations. This speedup is visualized in Figure 1b in the main text.

To this end, Figures 9–10 present intermediate frames extracted from four representative videos at five synchronized iteration checkpoints (∼100, 600, 1600, 4000, and ≥5000 iterations). The PSNR and iteration count are overlaid on each frame for reference. Across all examples, our method (top row) establishes coherent macroscopic structure within the first few hundred iterations, while Bézier Splatting (bottom row) exhibits a noisy "polygon-soup" distribution that slowly resolves through aggressive local edits. This visually supports our claim: the speedup originates from a better optimization trajectory enabled by progressive stratification, not merely from per-step efficiency.

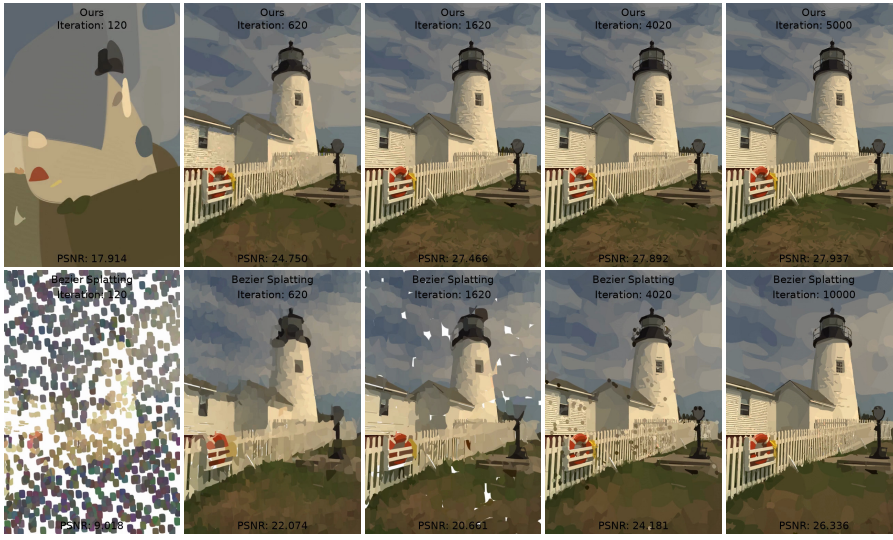

Fig. 11: Optimization trajectory on Kodak `kodim19` (portrait). Top: ours; bottom: Bézier Splatting at matched iterations. Our hierarchical refinement quickly converges to clean silhouettes, while the baseline keeps scattered fragments around the boundaries throughout training.

Fig. 12: Optimization trajectory on DIV2K `0112`. Top: ours; bottom: Bézier Splatting. The portrait scene benefits the most from progressive stratification — skin tones and fabric shading are recovered smoothly in our method, while the baseline distributes high-frequency noise across the face throughout training.