Segmentation-Based Parametric Painting

Supplementary Material

(a) Expressionist (b) Painterly (c) Abstract

Figure 10. Styles: (a) Expressionist, consisting of thick, textured, loose and low fidelity representation where subject matter is not a priority. (b) Painterly, think, shaped strokes with a loose touch, enough to balance the scene recognition and not yield a high-fidelity painting style, (c) Abstract, scattered, fewer strokes that capture the essence of the scene.

	Realistic	Painterly	Abstract Expressionist	Abstract
Num. Passes (P)	4	4	1	2
Num. layers (K)	1	var	var	var
Num. Strokes (T)	[36, 49, 64, 81]	[9, 16, 16, 9]	[9]	[9, 16]
Num. DAMs (V)	$(H//128 * W//128)$	[25, 25, 50]	$(H//128 * W//128)$	[25, 25]
DAMs mode	U	U if $p = 1$ else S	U	S

Table 5. Style hyperparameters by default. Users can fine-tune these parameters for further control over the desire styles.

Paintings at Large Resolution

More paintings at large resolution are shown at the end of this document.

6. Optimization Method

This section provides further implementation details of our method. The pseudocode is shown in Algorithm 2. We establish the nomenclature as follows:

- Input image X, canvas C
- Number of painting passes P (int), from 1 to P
- Number of painting layers K (int), from 0 to $K-1$
- Number of Dynamic Attention Maps (DAMs), V (int). If uniform, DAMs follow an array of 128×128 patches uniformly distributed on image X and canvas C. $V = (H//128 * W//128)$
- Number of stroke parameters, T (int), per DAM, different per p, and can be different per k
- Semantic Segmentation Network, W, determines K

6.1. Implementation Details

For all styles, we use Adam optimizer [18] with a learning rate of 0.0002 and betas 0.5 and 0.99. We optimize all painting layers for 300 iterations, and set the canvas background color to black. For our semantic segmentation network, we use the DETR model [2], available at Hugging Face, which consists of a CNN (ResNet) backbone followed by an encoder-decoder Transformer, trained on the COCO Panoptic Segmentation task.

6.2. Style Parameters

Our method takes in an image X and a style input as a string between "realistic", "painterly", "abstract", and "expressionist". Note that our optimization method does not apply transfer style techniques. Rather, we adjust optimization hyper-parameters such as the amount of dynamic attention maps (DAMs), stroke budget, or painting passes, to achieve different style variations. We found the combination of such parameters empirically, and we keep the parameters shown at Table 5 as default. However, users can change it to fine-tune the paintings to further adapt the painting. All styles follow a coarse-to-fine strategy, halving the stroke thickness following the formula: $a_p = 2^{1-p}a_1$, where $a_1 = 0.8 * 128$, that is, the 80% of the length of a 128×128 patch. The set of segmentation layers $\{k\}_0^{K-1}$ is generated by a segmentation network W, such that $\{k\} = W(X)$.

CVPR
#8699

Realistic Style Our realistic style is generated by setting $P = 4$, $K = 1$, setting DAMs to uniform, and thus $V = (H//128 * W//128)$. The stroke budget T on each patch follows $s_p = (p + 5)^2$.

Painterly Style The painterly style is generated by setting $P = 4$, applying a first foundational painting pass consisting of uniform DAMs ($V = (H//128 * W//128)$) and no segmentation distinction ($K_p = 1$ if $p = 1$). For $p > 1$, we activate segmentation layers, that is, $K_p = |W(X)|$, where X is the input image and $|\cdot|$ is the number of the segmentation layers; and switch to selective DAMs. The stroke budget T on each patch is limited to 16 except for the first and last passes which is 9.

Abstract Styles Within the abstract style, our method can yield different degrees of abstraction. For simplicity, we grouped them together in the main paper. However, there are two fundamental differences in the way they work. Figure 10 shows the two abstract styles. The expressionist style (a) is captured by the use of loose, efficiently optimized thick strokes with heavy texture. This produces a painting that prioritizes the effect and ambient of the scene by not caring too much about accurate shapes. The abstract style (c) is a looser representation than the painterly style, yet capturing shapes and subject matter better than the expressionistic abstract style.

Given a segmentation network W, the expressionist style is generated by setting DAMs to uniform, stroke budget $T = 9$, $P = 1$, $K = |W(X)|$, where X is the input image and $|\cdot|$ is the number of the segmentation layers. The abstract style is achieved by setting DAMs to selective, $P = 2$, stroke budget $T = [9, 16]$, $V = [25, 25]$, and $K = |W(X)|$.

6.3. Stroke Parameterization and Differentiable Renderer

We use the differentiable renderer provided by [15], which consists of 3 fully connected layers of dimensionality 512, followed by convolutional layers, and a sigmoid nonlinearity function to obtain a $1 \times 128 \times 128$ patch. A diagram of the architecture is shown in Figure 11.

Each stroke is formally represented as $\boldsymbol{s_t} = (x_0, y_0, x_1, y_1, x_2, y_2, r_0, r_2, t_0, t_1, R, G, B)$, where (x, y) are the Cartesian coordinates of the curve's control points, (r) denotes the radii at the endpoints, and (t) indicates the transparency levels. Typically, the transparency variables are set to unity, rendering all strokes opaque.

In the rendering pipeline, given a stroke budget of T, the parameters are either randomly or semi-randomly initialized in the configuration space of dimensions $T \times 13$. Subsequently, these parameters are fed into the differentiable rendering function G, which produces an array of T rasterized alpha strokes with dimensions 128×128. The color channels are obtained by element-wise multiplication of the alpha stroke with the respective RGB values. Each

Algorithm 2 Pseudo-code of our method.

procedure PAINT($imagePath$, $style$)
  $painter \leftarrow$ INITIALIZEPAINTER($imagePath, style$)
  $Segment.LayersK \leftarrow W(imagePath)$
  for $p \leftarrow 1$ to P **do** ▷ Painting passes
    for $k \leftarrow 0$ to $K - 1$ **do** ▷ Painting layers
      $DAMs \leftarrow$ GETDAMS($mode, p, |V|$)
      $canvas_p \leftarrow$ GETCANVAS($DAMs, p$)
      $strokes \leftarrow$ INITSTROKES($DAMs, T, p$)
      $strokes \leftarrow$ OPTIMIZE($strokes, canvas, style, k$)
      $canvas_p \leftarrow$ UPDATECANVAS($canvas_p, strokes$)
      Append $canvas_p$ to $allCanvasesL$
    end for
  end for
  COMPOSEFINALPAINTING(canvas)
  return final painting
end procedure
procedure INITIALIZEPAINTER($imagePath$, $parameters$)
  Process input image and set up canvas
  Define renderer, segmentaiton network, and perceptual network
  return painter object
end procedure
procedure GETDAMS($mode, p, V$)
  Computes DAMs based on mode and max number V
  return image X crops and DAMs coordinates
end procedure
procedure GETCANVAS($DAMs, p$)
  Crops $canvas_p$ based on DAMs
  return canvas crops
end procedure
procedure INITSTROKES($DAMs, T$,p)
  Initializes random stroke parameters based on DAMS, T and p
  return Stroke parameters
end procedure
procedure OPTIMIZE($strokes, canvas, style, k$)
  Optimizes strokes based on style parameteres
  Perform rendering to compute loss
  return strokes
end procedure
procedure COMPOSEFINALPAINTING($canvas$)
  Merge layers and apply final adjustments to canvas
  Save final painting
end procedure

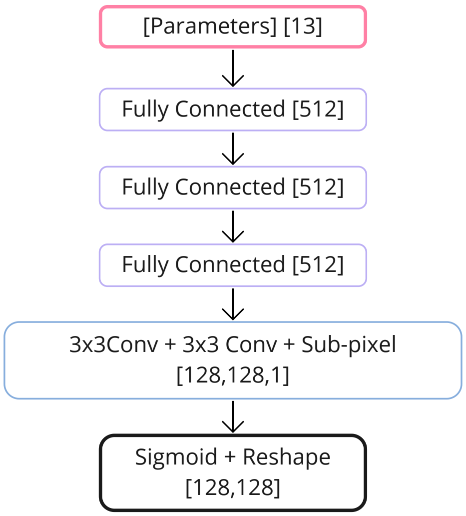


Figure 11. Neural differentiable renderer architecture provided by [15].

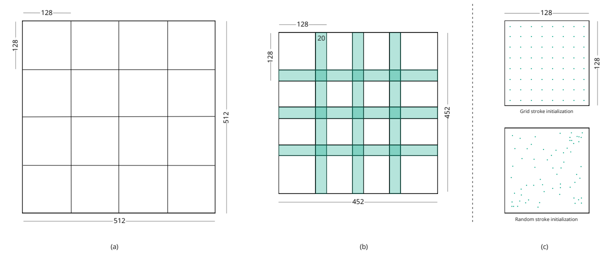


Figure 12. Canvas composition by patches. (a) Patches are organized into a structured grid without overlaps. (b) Example of an organization of patches in a structured grid with overlaps of 20 pixels. (c) Initialization of strokes before being optimized: strokes are evenly distributed across the x and y axis of the patch (upper part). Strokes are randomly sampled from a uniform distribution (bottom part).

rasterized alpha stroke is sequentially composited onto the canvas in accordance with the blending equation:

$$\text{canvas}_t = \text{canvas}_{t-1} \odot \text{alpha}_t + \text{stroke}_t \tag{3}$$

Here, t represents the temporal index within the stroke budget T, alpha_t refers to the alpha channel generated by the differentiable renderer G, and canvas_{t-1} denotes the canvas state prior to the incorporation of the new stroke. The initial canvas background is configured to be black across all experimental setups. The loss function $\mathcal{L}$ is computed between the reference image I and the finalized artwork C_T, followed by the backpropagation of the gradient for parameter optimization. This cycle iterates until the optimization algorithm converges to a stable state.

6.4. Impact of Stroke Initialization

Stroke initialization critically affects the algorithm's performance. In its most rudimentary form, the algorithm randomly initializes stroke parameters, uniformly distributed within the range $[0, 1]$. In a more advanced form, we can evenly distribute the total number of strokes along the two dimensions of the canvas in the form of a grid. This is done by anchoring the middle brushstroke coordinates to such grid, and then control the length of the stroke and the position of the start and end brushstroke coordinates using a Gaussian distribution. To illustrate, Figure 13 presents a series of examples showcasing paintings rendered at 128×128 pixels with different stroke initializations and stroke widths. These instances demonstrate that grid-based initialization, combined with Gaussian-controlled stroke lengths, substantially simplifies the optimization process. Intriguingly, although random stroke initialization may yield lower L_1 losses, it frequently results in less visually appealing outputs.

6.5. Patch Strategy and Stroke Initialization

Our method optimizes the stroke parameters of all N patches in the general canvas in batch. Once the strokes are optimized and all patches are painted using soft blending, we compose them back together with overlaps. For all experiments we use overlap of 20 pixels, as shown in Figure 12 (b).

We also perform an analysis on the computational cost in the number of patches and number of strokes. Figure 14 shows an analysis of the cost of increasing the number of patches (left) and increasing the stroke budget (right). Increasing resolution to 4x results in a cost of 1.56x. The bottleneck of SBR algorithms, regardless of whether we use patch-based strategies or not, is the number of strokes because of its linear relationship. That is, doubling the number of strokes results in double time to compute. This is because, generally, a blending operation is performed serially, instead of in batch. We perform this analysis on a single RTX A6000 Nvidia GPU.

6.6. Loss Ablation

We perform an ablation studies in loss functions. Generally, loss function, stroke model and number of strokes are responsible for the outcome of a painting. In this section, we analyze the impact of different loss function, see Figure 15. While L_1 works fine for reconstruction (top row), it

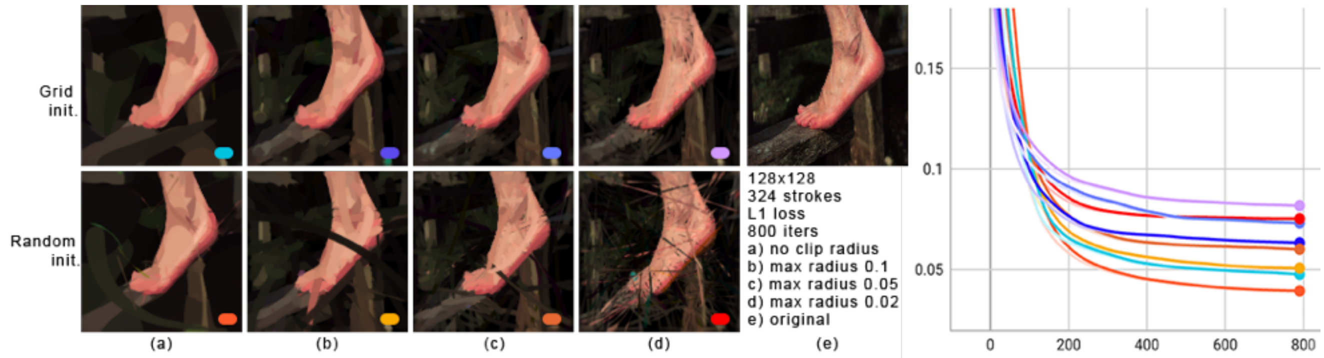


Figure 13. Difference of quality of paintings at 128x128 between random and grid stroke initialization. Top row corresponds to a grid stroke initialization, bottom row corresponds to random stroke initialization. Columns correspond to clipping the maximum stroke width the model is allowed to paint with: (a) no constraint, (b) maximum width 0.2*canvas size, (c) 0.1*canvas size and (d) 0.04*canvas size. All paintings use 324 strokes, and L1 as pixel loss function. We let the optimization run for 800 iterations.

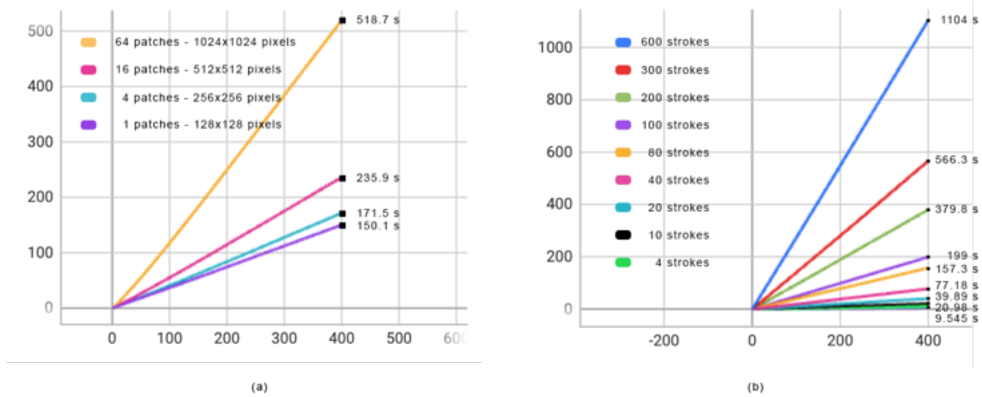


Figure 14. Patch and stroke cost. (a) Computational cost of painting by patches approach with a budget of 100 strokes per patch and 400 optimization steps. Each patch is 128x128. Increasing image size by double does not translate to a linear cost. (b) The computational cost of increasing stroke budget is approximately linear. Computed on a single RTX A6000 Nvidia GPU.

seems unable to capture finer details. Adding a perceptual loss helps alleviates this issue. We try using CLIP [29] as a type of perceptual loss, but it seems to add some noise to the painting. Second and fourth rows use a small weight for perceptual and CLIP losses, resulting in imperceptible contributions. However scaling up the weight to 0.1 seems to work better for reconstructions, being the perceptual loss in the third row the one that achieves best reconstructions. For realistic paintings, we choose a combination of L1 + perceptual loss, computed as shown in Section 3.3 in the main paper. CLIP loss is calculated as follows:

$$\mathcal{L} = (CLIP(I) - CLIP(C))^2 \tag{4}$$

where CLIP is the image encoder of the CLIP ViT-B/32 model.

7. Comparison with SOTA

We show more stroke distributions in Figure 16 across different subject matters. SNP and L2P present a more uniform distribution of strokes, barely differentiating between semantic areas or objects in the scene. While L2P presents a clear uniform distribution across the entire canvas, SNP is able to scatterly capture some regions of interest, but still suffer to bias the painting towards the main objects in the scene. PT does the absolute opposite, that is, it fixates a disproportionate amount of strokes to a specific area, which does not match semantic regions, while leaving the rest of the canvas with very few and big strokes. This is clearly visible in the picture at the middle row. The distribution is concentrated in a blob spanning the building and the pavement equally. The same event happens in the last row where the lake and the lavender flowers that are closer to the water

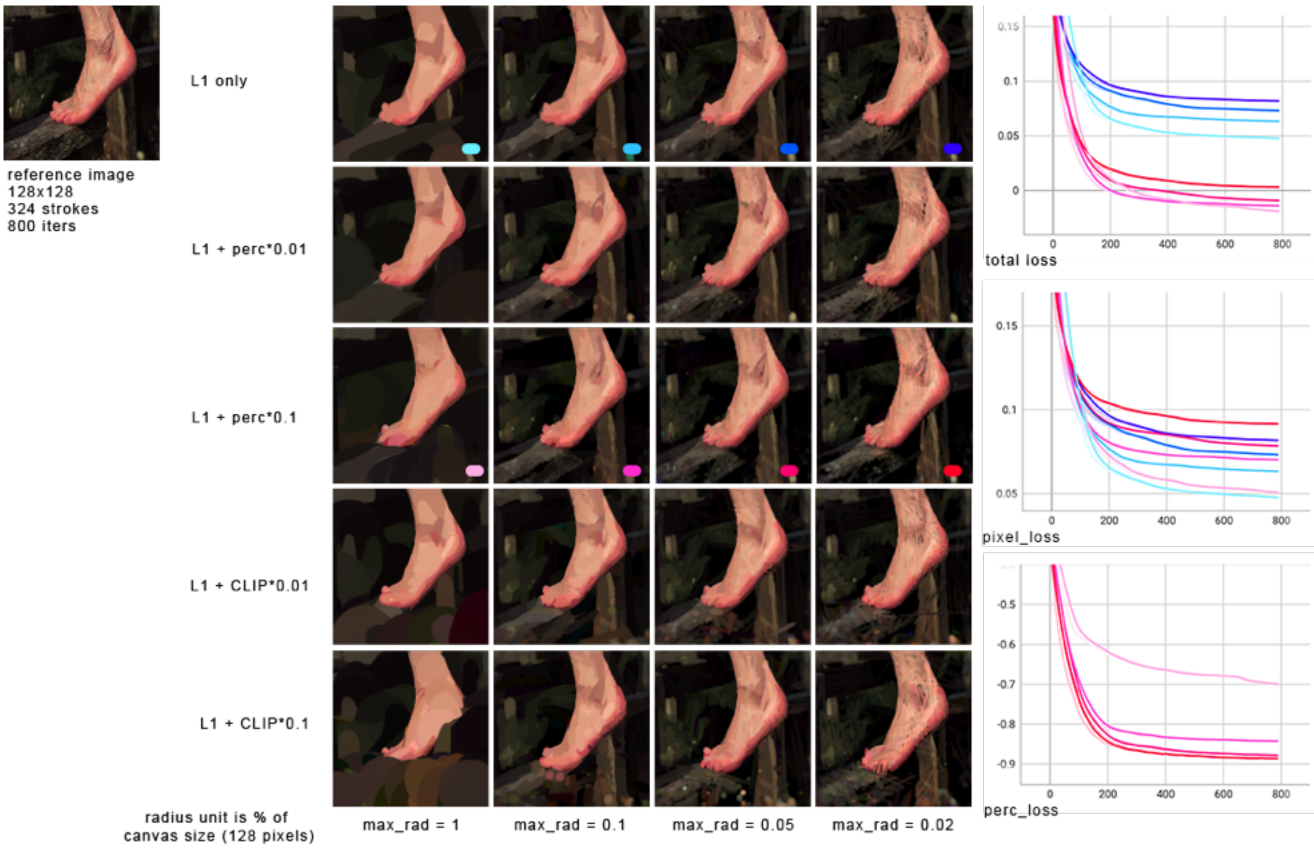


Figure 15. Difference in loss functions. We show the difference in the outcome when using L1, perceptual loss and CLIP loss. A weight of 0.01 seems very low for either perceptual or CLIP to make an impact. However, a weight of 0.1 makes the perceptual loss define some fine details than L1 loss alone cannot (see definition in the toes). CLIP loss introduces some noise in the painting.

receive a higher quantity of strokes.

Ours, however, sits in between L2P / SNP and PT. While focusing on a specific object in the picture, with semantic coherence, it is able to also distribute strokes on other important areas of the picture. For instance, the bottom row shows how flowers, lake and mountains are better captured, while leaving sky with fewer strokes. In the case of the building, strokes are concentrated around it, since this is clearly the most important semantic part of the image. However, our method is able to redirect its stroke distribution according to the user's desire as shown in Figure 17. In this figure, we show a painting dissection based on our semantic segmentation pipeline. Except for last row, each column shows emphasis on a different k-th layer. The second row shows a distribution of all the strokes in the painting, while the fifth shows a distribution of only the strokes pertaining to each k-th layer.

8. User Study

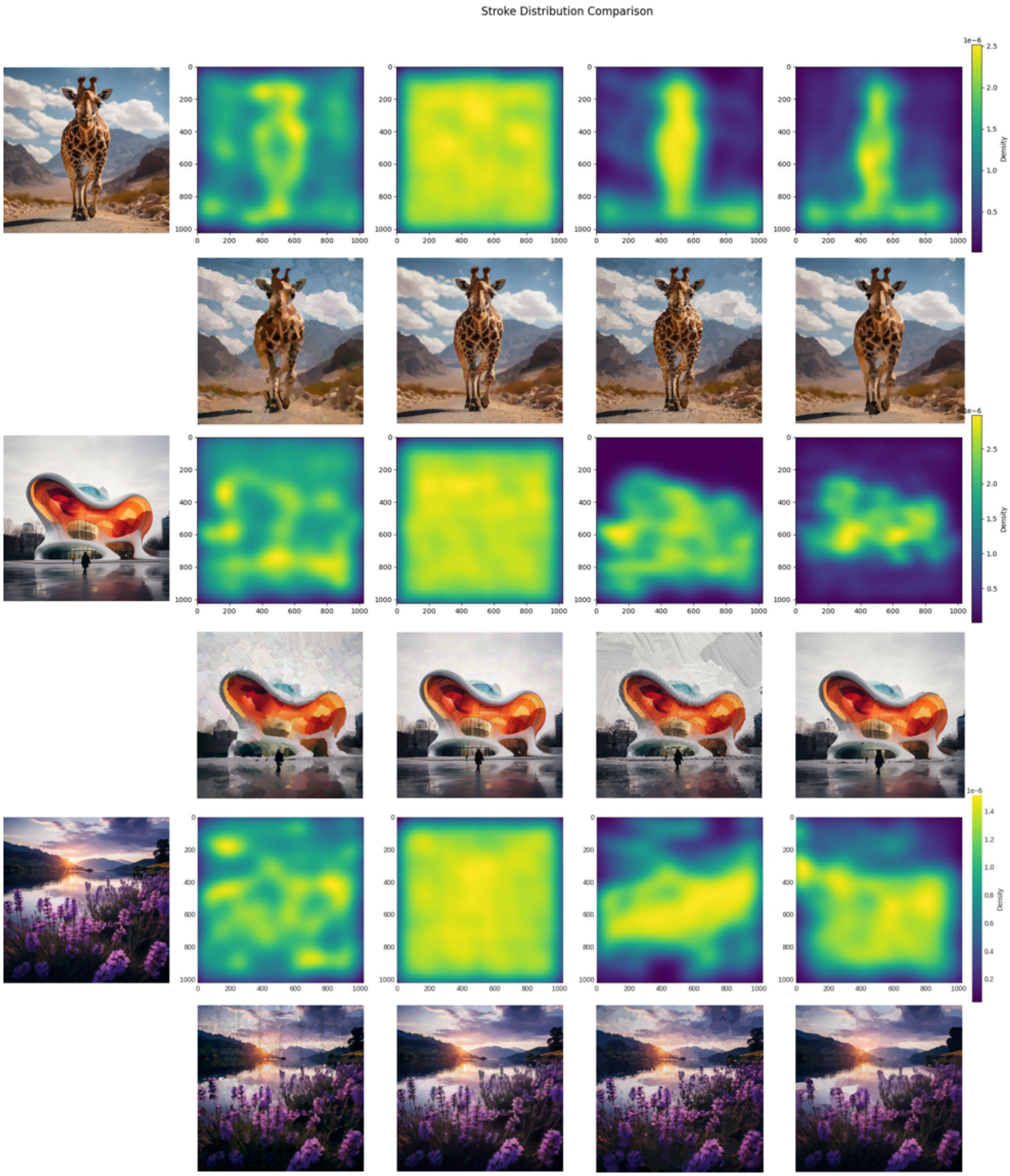
The user study is divided in two sections with 6 images each. The first section evaluates realistic paintings, and asks which painting is a better painting of an input image. The second section evaluates visual appeal, and asks participants to choose the painting that looks more natural and less computer made. In total, we have 12 two-way side by side comparisons. The participants see first an input photograph, and below the photograph we place a pair of paintings. We ask to choose left or right painting. We compare our paintings with three recent methods: an optimization-based method "Stylized Neural Painting" (SNP) [41], a learning-based method with Transformer "Paint Transformer" (PT) [23], and a RL method "Learning to Paint" (L2P) [15]. We randomize the order in which we show the pairings.

The complete user study is shown in Figures 24 to 26. All four pairs from figure Figure 24 and the first two rows in Figure 25 correspond to the first section of the user study. These pairings correspond to the left side of Table 2 in the main paper. IDs are Figure 24 top row: A5, second row:

CVPR
#8699

A3, third row: A4, fourth row: A6. In Figure 25, top row: A1, second row: A2. The last two rows of Figure 25 and Figure 26 belong to the second section of the study. They have the following IDs; Figure 25 third row: B1, fourth row: B5. Figure 26, from top to bottom: B3, B6, B2 and B4, respectively.

CVPR #8699



(a) Input (b) SNP (c) L2P (d) PT (e) Ours

Figure 16. Stroke distribution analysis: SNP and L2P present a uniform distribution across the canvas. L2p is completely scene-agnostic, and SNP cannot distinguish between semantic areas. PT presents a severe contrast between objects and background, and struggles to differentiate between semantic areas of the image. Ours distribute strokes based on semantic areas.

CVPR
#8699

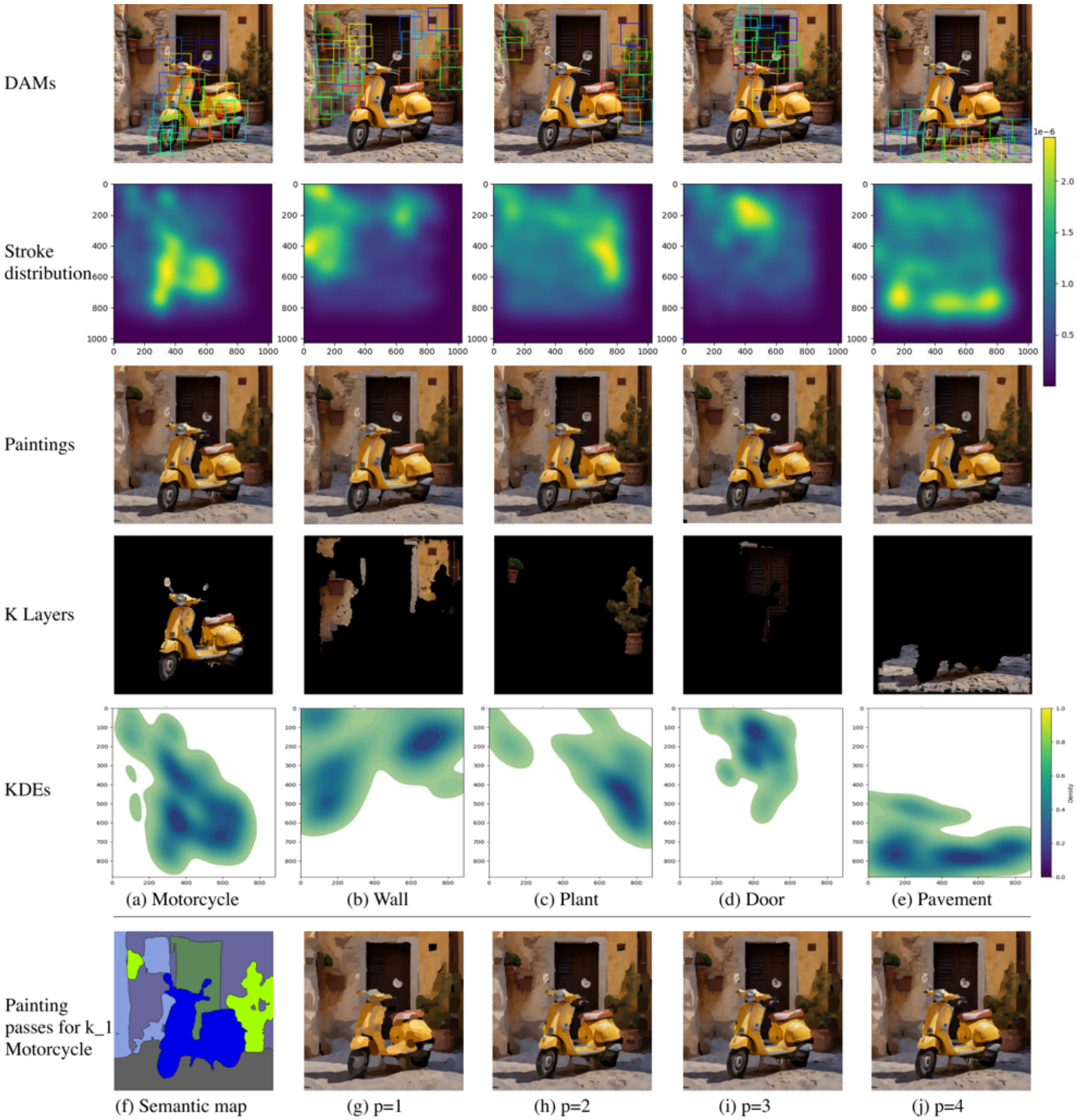


Figure 17. Painting analysis based on semantic layers and dynamic attention maps (DAMs). Top row shows DAMs per semantic layer k. Second row shows the overall stroke distribution over the canvas based on biasing the painting towards different semantic areas, and the third row shows the corresponding paintings. The fourth row shows each layer k independent from the rest of the painting. The fifth row shows the Kernel Density Estimation of the strokes that correspond to only such layer k. The bottom row shows the semantic map, and the four painting passes, from coarse to fine, with a bias on the vehicle over the rest of the scene.

(a) Ours - Realism

(b) L2P

Figure 18. Comparison of our method (a) against L2P [15] (b) on realistic images. Note how L2P paintings have visible seams and blurry areas.

Figure 19. Comparison of our method (a) against PT [23] (b) and SNP [41] (c) on painterly images.

CVPR
#8699

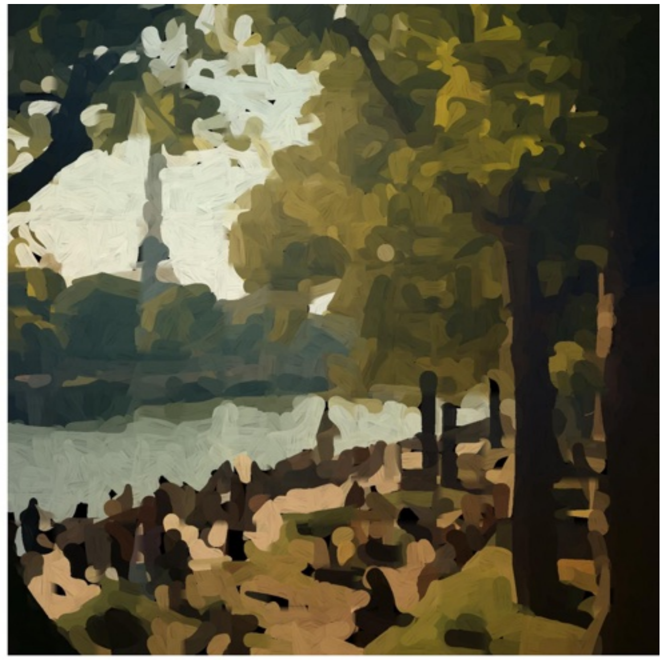
(a) Expressionist

(b) Painterly

(c) Realism

CVPR
#8699

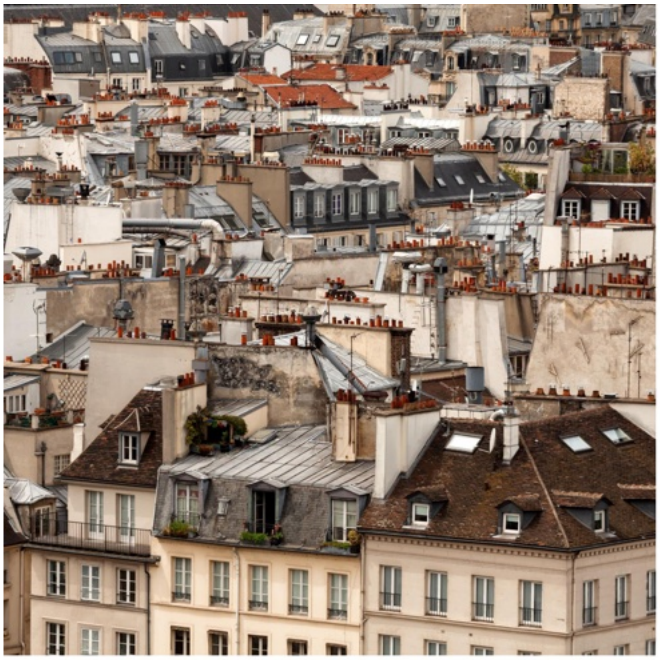
(a) Input



(b) Expressionist

(c) Painterly

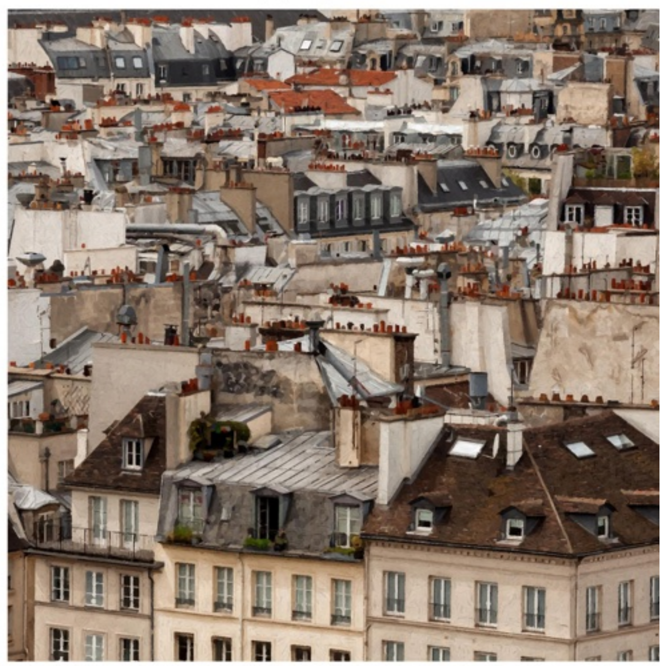
(d) Realism



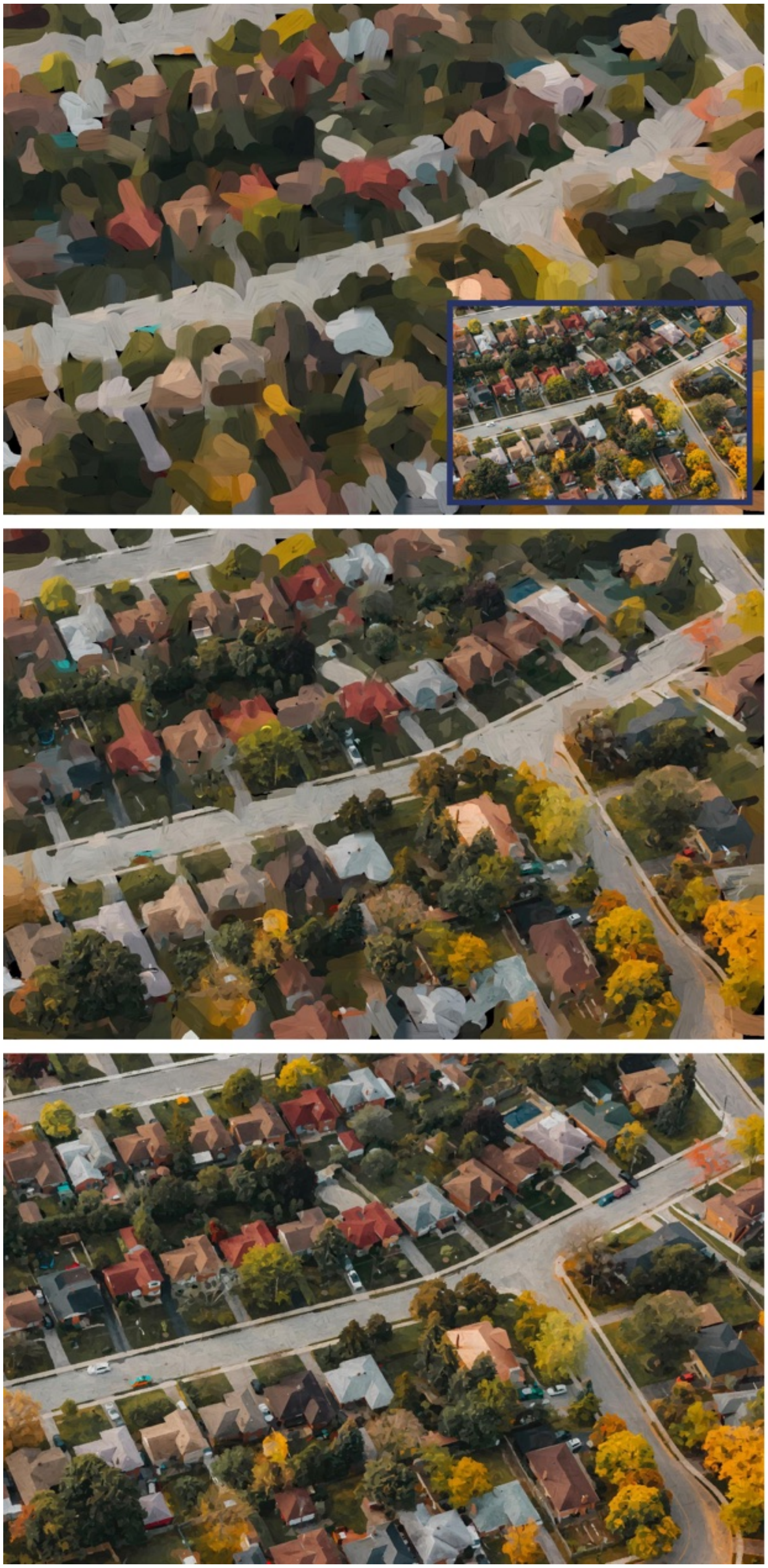
(a) Input

(b) Expressionist

(c) Painterly



(d) Realism



(a) Expressionist

(b) Painterly

(c) Realism

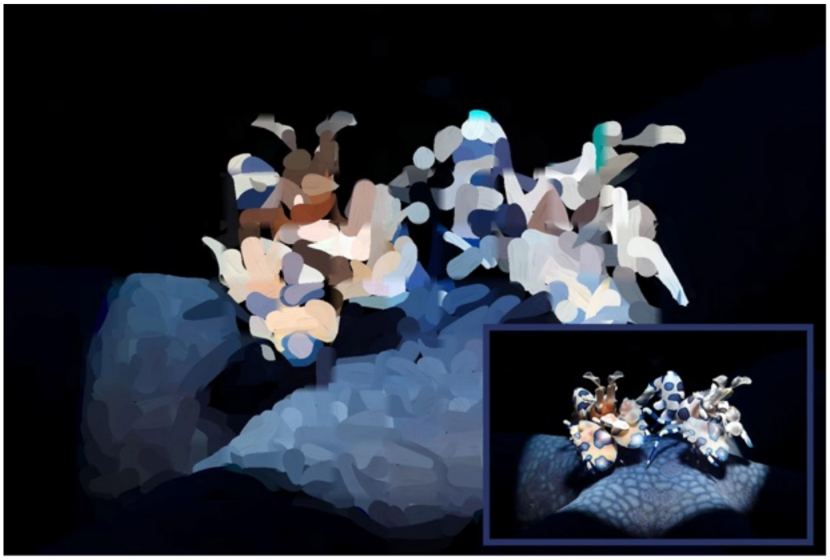
CVPR
#8699

(a) Expressionist

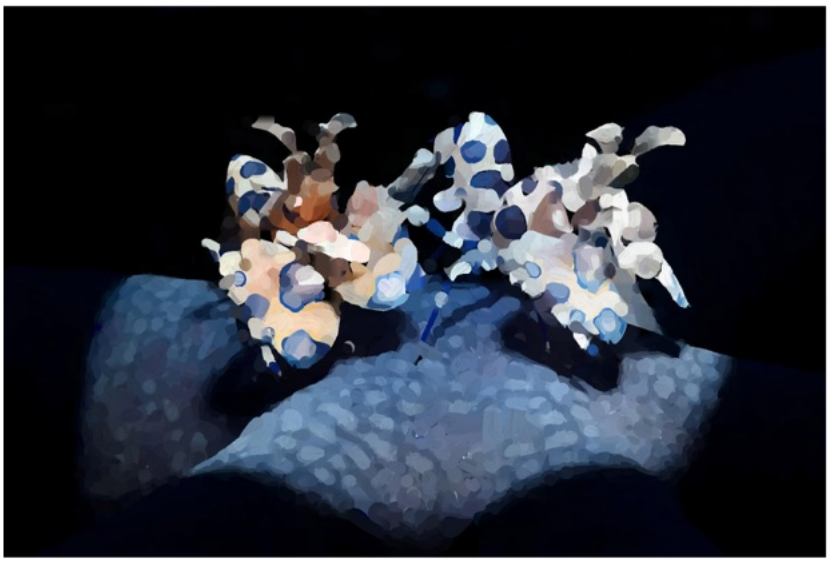
(b) Painterly

(c) Realism

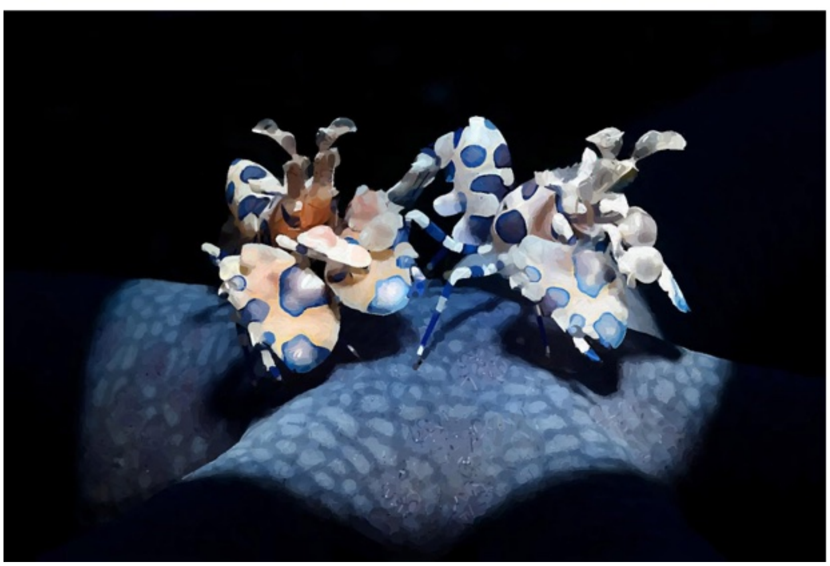
CVPR
#8699



(a) Abstract



(b) Painterly



(c) Realism


(a) Abstract


(b) Painterly

Coarse-to-fine (from left to right and top to bottom)

Figure 20. Painterly style with dynamic attention maps corresponding to the last 3 painting layers are shown at the bottom.

Figure 21. Painterly style with dynamic attention maps corresponding to the last 3 painting layers are shown at the left. Painting process is shown at the right.

Figure 22. Realistic Style (left) and painterly style (right). Dynamic attention maps for the last painting pass are shown as an inset.

Figure 23. Realistic painting process. Painted using uniform dynamic attention maps. From coarse (top left) to fine (bottom right)

Figure 24. Pairings shown in user study from section 1. IDs from top to bottom: A5, A3, A4, and A6

CVPR
#8699

Figure 25. Pairings shown in user study from section 1 and 2. IDs from top to bottom: A1, A2, B1, and B5

Figure 26. Pairings shown in user study from section 2. IDs from top to bottom: B3, B6, B2 and B4