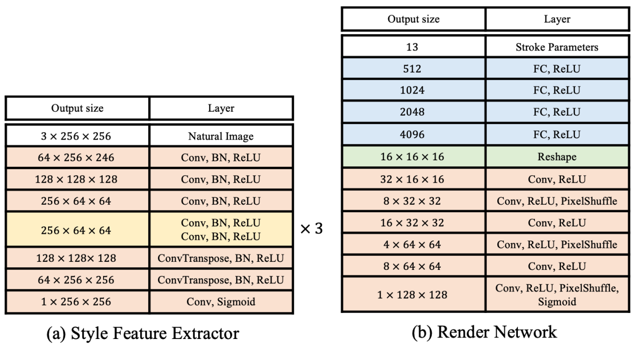
Learning Realistic Sketching: A Multi-Agent Reinforcement Learning Approach

Anonymous Authors

The supplementary material is organized into four sections. Initially, it provides a detailed description of the network structures discussed in Section 3 of the main text. It then explores the training details of our method corresponding to Section 3.5. Subsequently, it examines the specifics of the render network as outlined in Section 3.4. Finally, it offers additional analysis and results for Section 4.5.

1 NETWORK STRUCTURE



Output size	Layer
$3 \times 256 \times 256$	Natural Image
$64 \times 256 \times 246$	Conv, BN, ReLU
$128 \times 128 \times 128$	Conv, BN, ReLU
$256 \times 64 \times 64$	Conv, BN, ReLU
$256 \times 64 \times 64$	Conv, BN, ReLU Conv, BN, ReLU
$128 \times 128 \times 128$	ConvTranspose, BN, ReLU
$64 \times 256 \times 256$	ConvTranspose, BN, ReLU
$1 \times 256 \times 256$	Conv, Sigmoid

(a) Style Feature Extractor

Output size	Layer
13	Stroke Parameters
512	FC, ReLU
1024	FC, ReLU
2048	FC, ReLU
4096	FC, ReLU
$16 \times 16 \times 16$	Reshape
$32 \times 16 \times 16$	Conv, ReLU
$8 \times 32 \times 32$	Conv, ReLU, PixelShuffle
$16 \times 32 \times 32$	Conv, ReLU
$4 \times 64 \times 64$	Conv, ReLU, PixelShuffle
$8 \times 64 \times 64$	Conv, ReLU
$1 \times 128 \times 128$	Conv, ReLU, PixelShuffle, Sigmoid

(b) Render Network

Figure 1: The two neural network architectures are: (a) Style Feature Extractor and (b) Render Network.

1.1 Style Feature Extractor

As illustrated in Figure 1.a, we employ an encoder-decoder architecture for the Style feature extractor. It takes a natural image I as input and processes it through convolutional layers, two downsampling layers, three residual blocks, and two upsampling layers to extract the feature map F of the natural image. The decoder, serving as a generator, converts F into sketches. It achieves this transformation through and a convolutional layer, outputting the generated sketch $\tilde{I}$.

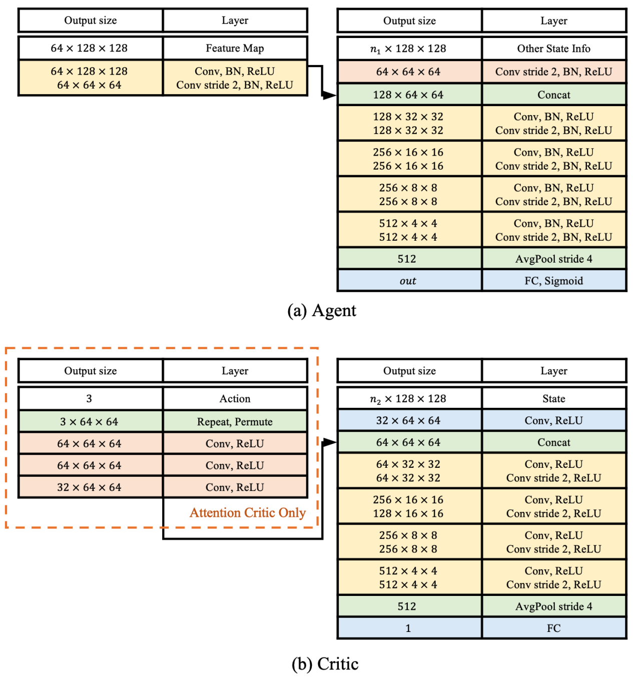
1.2 Render Network

The renderer network achieves the mapping from stroke parameters and the current canvas to the subsequent canvas with the added strokes. As illustrated in Figure 1.b, the network receives stroke parameters a_t^D, expands their dimensions to 4096 through four fully connected layers, and reshapes them to $16 \times 16 \times 16$. It then processes this data through convolutional layers and PixelShuffle operations to output a $1 \times 128 \times 128$ rendered image.

1.3 Agent

As shown in Figure 2.a, for both agents, the networks receive inputs from two sources that are combined via a concatenation operation. The original feature map's shape is $64 \times 256 \times 256$. For the attention agent, the state information channel count $n1$ is set to 8, and the feature map is resized to $64 \times 128 \times 128$ through interpolation. For the drawing agent, the state information channel count $n1$ is set to 4, and the input feature map undergoes ROI Align to crop and resize the features of the drawing area to $64 \times 128 \times 128$.

The merged feature is subsequently processed through residual blocks and a fully connected network to generate actions. Considering the high variability and complexity of real-world images, the agents benefit from implementing batch normalization. The attention agent outputs the size and position of the next drawing area a^A, with an output dimension of 3. Meanwhile, the drawing agent outputs stroke parameters a^D, with an output dimension of 13.



Output size	Layer
$64 \times 128 \times 128$	Feature Map
$64 \times 128 \times 128$ $64 \times 64 \times 64$	Conv, BN, ReLU Conv stride 2, BN, ReLU

Output size	Layer
$n_1 \times 128 \times 128$	Other State Info
$64 \times 64 \times 64$	Conv stride 2, BN, ReLU
$128 \times 64 \times 64$	Concat
$128 \times 32 \times 32$ $128 \times 32 \times 32$	Conv, BN, ReLU Conv stride 2, BN, ReLU
$256 \times 16 \times 16$ $256 \times 16 \times 16$	Conv, BN, ReLU Conv stride 2, BN, ReLU
$256 \times 8 \times 8$ $256 \times 8 \times 8$	Conv, BN, ReLU Conv stride 2, BN, ReLU
$512 \times 4 \times 4$ $512 \times 4 \times 4$	Conv, BN, ReLU Conv stride 2, BN, ReLU
512	AvgPool stride 4
out	FC, Sigmoid

(a) Agent

Output size	Layer
3	Action
$3 \times 64 \times 64$	Repeat, Permute
$64 \times 64 \times 64$	Conv, ReLU
$64 \times 64 \times 64$	Conv, ReLU
$32 \times 64 \times 64$	Conv, ReLU

Output size	Layer
$n_2 \times 128 \times 128$	State
$32 \times 64 \times 64$	Conv, ReLU
$64 \times 64 \times 64$	Concat
$64 \times 32 \times 32$ $64 \times 32 \times 32$	Conv, ReLU Conv stride 2, ReLU
$256 \times 16 \times 16$ $128 \times 16 \times 16$	Conv, ReLU Conv stride 2, ReLU
$256 \times 8 \times 8$ $256 \times 8 \times 8$	Conv, ReLU Conv stride 2, ReLU
$512 \times 4 \times 4$ $512 \times 4 \times 4$	Conv, ReLU Conv stride 2, ReLU
512	AvgPool stride 4
1	FC

(b) Critic

Figure 2: The two neural network architectures are: (a) Agent and (b) Critic.

1.4 Critic

The critic, a crucial component of the TD3 algorithm, predicts the value Q of the state-action, which assesses the expected cumulative reward of the current state and the agent's action. As depicted in Figure 2.b, for the attention module, the critic receives both state and action inputs, where the state channel n_2 is 8, comprising (C_t, I^{gt}, M_t, t). These inputs are concatenated and processed through four residual blocks, concluding with an average pooling layer that feeds into a fully connected network to compute the value function Q.

In the drawing module, the critic only receives state inputs with the channel count n_2 set to 10, which includes $(C_t, C_t + 1, I^{gt}, t)$, and outputs the value V of the current state.

2 TRAINING DETAILS

In the training of the attention agent, we adopt a model-free approach due to the non-differentiable nature of the Paste operation, which makes the state transition function non-differentiable. The critic is to estimate the expected reward for the agent's action a_t at state s_t, utilizing the Bellman equation as used in Q-learning:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma Q(s_{t+1}, \pi(s_{t+1})), \tag{1}$$

where $r(s_t, a_t)$ represents the reward received from the environment for executing action a_t at state s_t. Meanwhile, the agent π^A is optimized to maximize the estimated $Q(s_t, \pi(s_t))$ by the critic.

In the training of the drawing agent, we employ a model-based approach because the render network is differentiable, allowing the state transition function trans to be differentiable as well. The critic inputs s_{t+1} instead of both s_t and a_t, and predicts the expected reward for the state. This leads to a new expected reward formulation, a value function $V(s_t)$, trained using discounted rewards:

$$V(s_t) = r(s_t, a_t) + \gamma V(s_{t+1}). \tag{2}$$

Here, the agent π^D is optimized to maximize $r(s_t, \pi(s_t)) + V(\text{trans}(s_t, \pi(s_t)))$ because the transition function $s_{t+1} = \text{trans}(s_t, a_t)$ is differentiable.

3 RENDER NETWORK

Traditional rendering processes are non-differentiable due to discrete rasterization operations, but neural renderers bridge this gap by leveraging the inherent differentiability of deep neural networks. Stroke renderers focus on mapping stroke parameters to canvas images rather than transforming 3D models into 2D images. To train effectively, they require differentiable renderers, enhancing painting quality and convergence speed. Even if rendering environments produce excellent results, interacting with non-differentiable painting environments deprives the agent of explicit feedback.

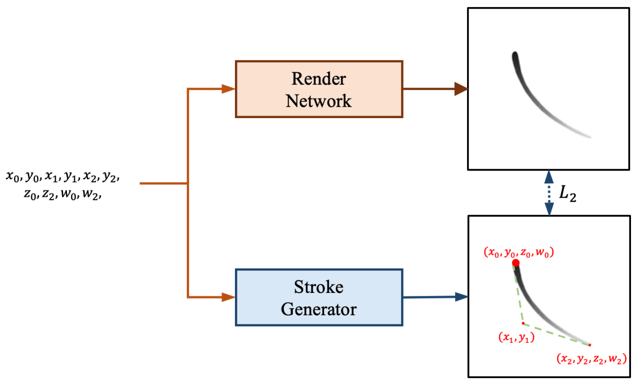


Figure 3: The architecture of the render network training.

As shown in Figure 3, during training, samples are generated by the stroke generator program and trained through supervised learning. Initially, stroke parameters are randomly sampled and rendered on fixed-size white images, simulating a painter's process. These stroke parameters are then passed to the render network to capture complex relationships between input parameters. The network outputs generated images, aiming to resemble real images. The L_2 distance measures the similarity between generated and rendered images, guiding parameter updates through backpropagation to optimize network weights and reduce the distance.

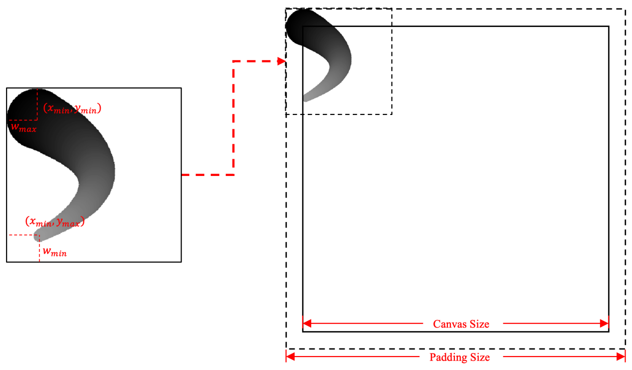


Figure 4: The left side shows the image rendered with stroke parameters at boundary values, pasted back to the edge of the overall canvas to ensure stroke continuity.

To ensure the continuity of the strokes, we constrain the positions and thickness of control points within the canvas in the Stroke Generator. As shown in Figure 4, we restrict the maximum and minimum values of the x, y coordinates and the width w of the control points to ensure strokes remain within the drawing area. Because these constraints prevent strokes from reaching the canvas edges, we apply padding to the canvas to ensure accurate depiction of edge positions.

4 ADDITIONAL ANALYSIS AND RESULTS

Table 3 of the main text reveals that with 50 strokes, the VSketch method, which employs VGG perceptual loss, ranks second only to ours in terms of PSNR and SSIM. However, it underperforms visually. This issue stems from VSketch's failure to preserve detailed integrity, resulting in significant blank areas. Paradoxically, these blank areas bring it closer to the ground truth at the pixel level than other baselines. RST employs SLIC to segment images for stroke initialization, frequently exceeding the predetermined number of strokes and necessitating pixel-level post-processing, which can disrupt the painting process. Consequently, as demonstrated in Figure 6 of the main text, RST with 50 strokes yields textures and details that surpass typical results for that stroke count.

Figure 5 further demonstrates the comparison between our method and baselines under 1000 strokes, while Figure 6 illustrates the intermediate stages of our sketching process, which align with human sketching habits. Initially, our method sets up the general structure and then progressively refines the details in subsequent stages. Additionally, supplementary materials include a video titled "sequential_realistic_ sketching.mp4," showcasing our method's sketching process from start to finish under a scenario of 1000 strokes.

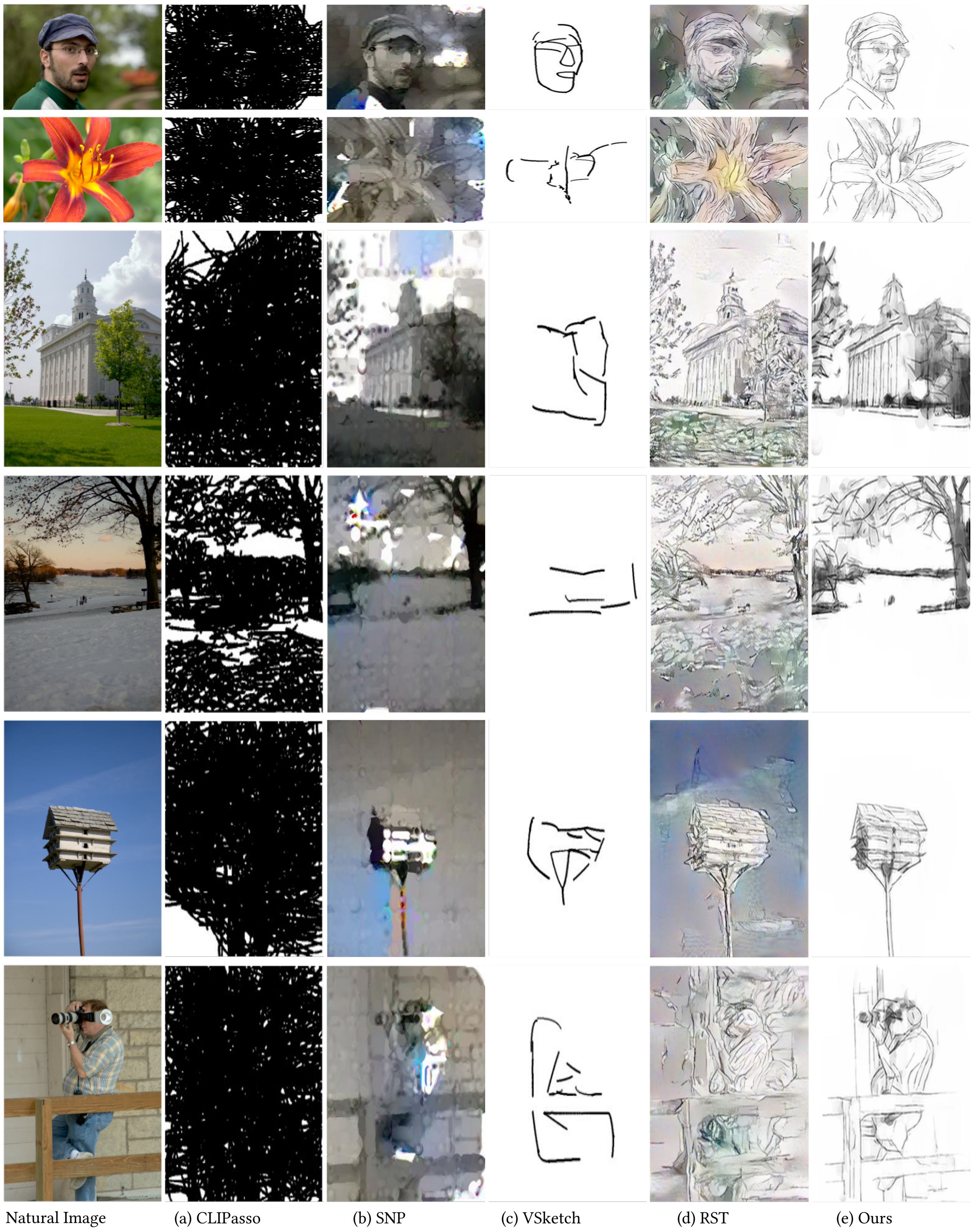


Figure 5: Additional comparison of existing drawing methods under scenarios with 1000 strokes.

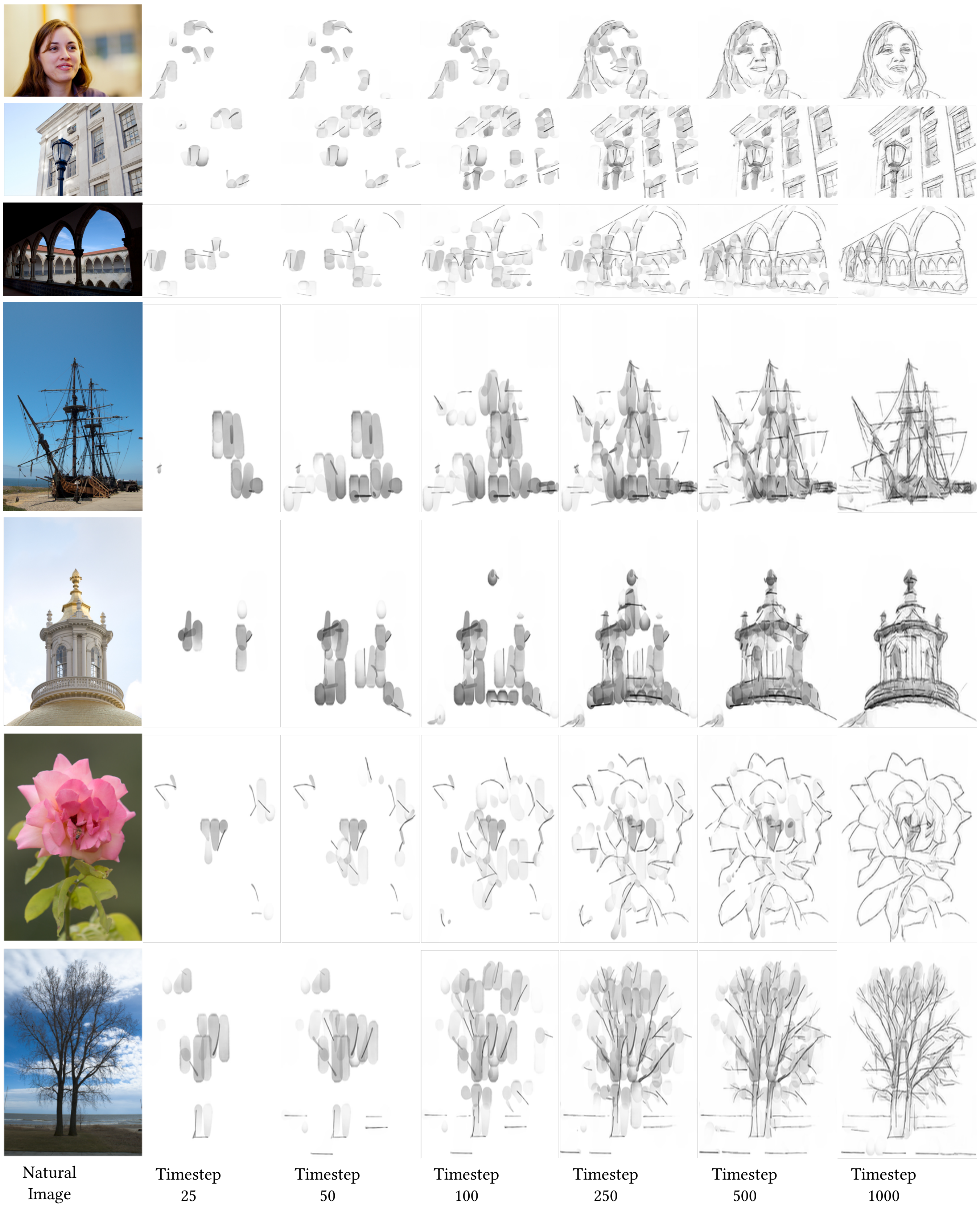


Figure 6: Drawing progression over a series of timesteps.