IEICE Transactions on Information and Systems,
vol.E96-D, no.5, pp.1134-1144, 2013.

Artist Agent: A Reinforcement Learning Approach to Automatic Stroke Generation in Oriental Ink Painting

Ning Xie
Tokyo Institute of Technology, Japan.
xie@sg.cs.titech.ac.jp

Hirotaka Hachiya
Tokyo Institute of Technology, Japan.
hachiya@sg.cs.titech.ac.jp

Masashi Sugiyama
Tokyo Institute of Technology, Japan.
sugi@cs.titech.ac.jp http://sugiyama-www.cs.titech.ac.jp/~sugi

Abstract

Oriental ink painting, called *Sumi-e*, is one of the most distinctive painting styles and has attracted artists around the world. Major challenges in Sumi-e simulation are to abstract complex scene information and reproduce smooth and natural brush strokes. To automatically generate such strokes, we propose to model the brush as a reinforcement learning agent, and let the agent learn the desired brush-trajectories by maximizing the sum of rewards in the policy search framework. To achieve better performance, we provide elaborate design of actions, states, and rewards specifically tailored for a Sumi-e agent. The effectiveness of our proposed approach is demonstrated through experiments on Sumi-e simulation.

Keywords

painterly rendering, stroke-based rendering, reinforcement learning, policy gradient.

1 Introduction

Among various techniques of *non-photorealistic rendering* [7], *stroke-based painterly rendering* synthesizes an image from a source image in a desired painting style by placing discrete strokes [10]. Such an algorithm simulates the common practice of human painters who create paintings with brush strokes. In this paper, we focus on automatic stroke generation in oriental ink painting.

Unlike western painting styles such as water-color, pastel, and oil painting, all of which overlap strokes into multiple layers [9, 17], oriental ink painting uses a few expressive strokes produced by soft brush tufts to convey significant information about a target scene. The appearance of the stroke is therefore determined by the shape of the object to paint, the path and posture of the brush, and the distribution of pigments in the brush.

Drawing smooth and natural strokes in arbitrary shapes is challenging since an optimal brush trajectory and the posture of a brush footprint[1] are different for each shape. Existing methods can efficiently map brush texture by deformation onto a user-given trajectory line or the shape of a target stroke [2, 8, 9, 14]. However, the geometrical process of morphing the entire texture of a brush stroke into the target shape leads to undesirable effects such as the undesirable folding and creased appearance at corners or curves.

To overcome this critical weakness, a more systematic approach was introduced recently [22]. In that paper, the problem of drawing brush strokes was formulated as a multi-step decision making process to minimize an accumulated energy of moving the brush. The *dynamic programming* (DP) method was used to obtain optimal brush strokes. It was demonstrated that smooth and natural brush strokes could be obtained by minimizing the accumulated energy.

However, DP is a model-based method and thus a stroke optimized by DP for a specific shape cannot be applied to other shapes even when the difference is small. Thus, the DP-based approach is not efficient if the target object is composed of many basic shapes, e.g., a Chinese character, since the optimal brush stroke for *each* shape has to be obtained. Furthermore, ordinary DP cannot directly handle continuous actions and states. Thus, the smoothness of brush strokes is highly dependent on the discretization of spaces.

The objective of this paper is to overcome the above weakness of the DP method. More specifically, we model a soft-tuft brush as an intelligent agent. Given any closed contour that represents the shape of a desired single stroke without overlap, the goal of our agent is to be able to draw a stroke by moving the brush on the canvas to fill in the given shape once from a start point to an end point on the contour with stable poses along a smooth continuous movement trajectory. MDP is a standard mathematical formalization of sequential decision making that has been successfully employed in many applications [13]. In this MDP framework, we let our brush agent learn which direction to move and how to keep the stable posture while sweeping over an arbitrary stroke shape (see Figure 1).

Reinforcement learning (RL) helps build an intelligent agent in an unknown environment [20]. This advantage motivates us to develop a method to optimize the brush agent's behavior of stroke drawing in the RL framework: First, we let the agent learn a desired drawing policy by maximizing the sum of rewards from a number of typical training shapes. Then, the agent applies the trained policy to drawing strokes for various new shapes.

The main challenges in this paper are two folds:

(i) The design of the brush agent. Our key idea is to design the state space of the brush

[1]We use a *footprint* to denote the region of a canvas which a brush stamps on.

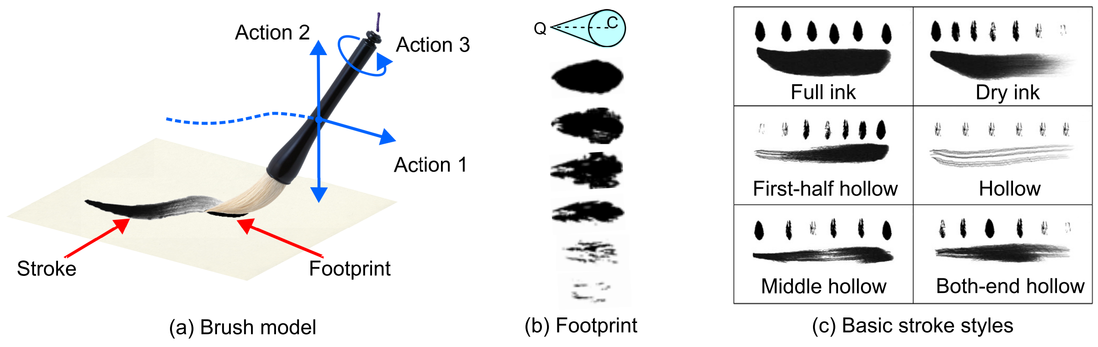


Figure 1: Illustration of our brush agent and its path. (a) In our model, a stroke is generated by moving the brush with the following 3 actions: Action 1 is regulating the direction of the brush movement, Action 2 is pushing down/lifting up the brush, and Action 3 is rotating the brush handle. In our system, only Action 1 is determined by reinforcement learning, and Actions 2 and 3 are automatically determined based on Action 1. (b) The top symbol illustrates our brush agent, which consists of a tip Q and a circle with center C and radius r. Others illustrate footprints of a real brush with different ink quantities. (c) There are 6 basic stroke styles: full ink, dry ink, first-half hollow, hollow, middle hollow, and both-end hollow. Small footprints on the top of each stroke show the interpolation order.

agent to be *relative* to its surrounding shape [19], e.g., boundaries and the medial axis. This allows the agent to learn a general drawing policy that is *independent* of the overall shape.

(ii) The design of a training strategy for the brush agent. We use the RL method called the *policy gradient* method [21] for agent training. An advantage of the policy gradient method over other RL methods is that it can naturally handle *continuous* states and actions, which are particularly important in the current paper to obtain smooth and natural brush strokes. To further improve the generalization ability of the agent to new shapes, we train the agent with *partial* shapes, by which the number and variation of training samples can be significantly increased. Another merit of using partial shapes is that even when the entire profile of a new shape is quite different from those of training data, the new shape may contain similar partial shapes and thus better generalization ability can be expected.

The rest of this paper is structured as follows. Section 2 gives a brief review of existing methods for generating brush drawings. Section 3 formulates the problem of automatic stroke generation as reinforcement learning and reviews the policy gradient method. Section 4 gives our specific design of states, actions, and rewards for automatic stroke generation, as well as the design of the training session. Section 5 shows experimental results and Section 6 concludes.

2 Related Works

In this section, we briefly review existing methods for generating brush drawings, which can be categorized into two approaches: *Physics-based painting* and *stroke-based rendering.*

2.1 Physics-Based Painting

Physics-based painting aims at simulating a real painting process and giving users intuitive and natural feeling when holding a mouse or a pen-like device. Several previous works modeled the brush shapes, its dynamics, and its interaction with the paper, as well as simulated the ink dispersion and absorption by a paper, e.g, the hairy brushes [18] and the physics-based models [16, 5, 4].

For interactive use, these virtual brushes are convenient to draw various styles of strokes. Although there exists extensive research literature along this line, automatically controlling a virtual brush with six degrees of freedom—three for the Cartesian coordinates and three for their angular orientation (pitch, roll, and yaw)—in addition to the dynamics of the tufts is highly complex and existing physics-based models are in fact simplifications of the real process.

Another major problem of the physics-based method is that the computational cost is usually very high for achieving satisfactory visual effects to human eyes. Some of them use *graphics processing units* for speedup [4]. However, due to over-simplification, none of these methods has been able to simulate certain special brush strokes such as *impasto* created with paint knives.

2.2 Stroke-Based Rendering

If a user has no painting expertise and is interested only in painting results rather than the painting process itself, stroke-based rendering is practically more desirable than physics-based painting.

The skeleton stroke method [11] generates brush strokes from two-dimensional paths given either through user interaction or by automatic extraction from real images. However, specifying and varying the width and texture of a stroke along a given path is practically very difficult. One of the solutions to this problem is to specify a *stroke backbone* [8] manually by a user. However, its limitation is that appropriately tuning parameters for each control point is difficult and time-consuming.

3 Reinforcement Learning Approach to Automatic Stroke Generation

In this section, we formulate the problem of automatic stroke generation as a Markov decision process and review the policy gradient method.

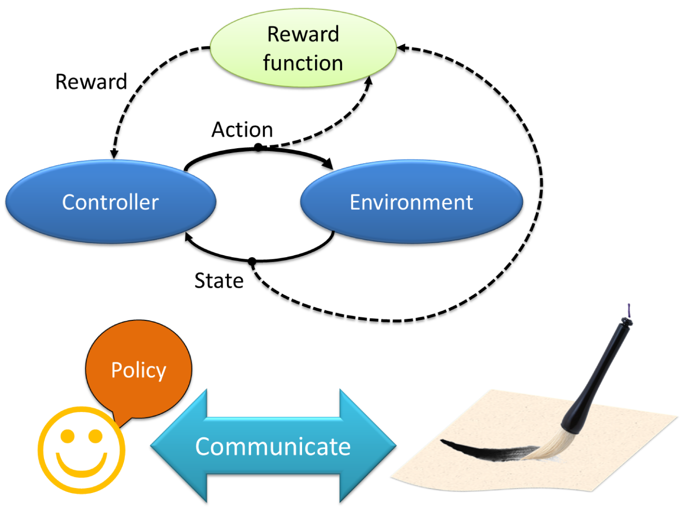


Figure 2: The flow of interaction between the agent and the environment.

3.1 Overview

A boundary map of an image can provide valuable information for various image analysis and interpretation tasks such as segmentation and object description [12]. In our stroke generation task, the boundary map represents all segments of expected strokes. Here, we treat the boundary map as an input to our brush agent for drawing strokes inside the boundaries.

We use the following terms in our formulation (see Figure 2):

- A *controller* is a decision-making algorithm of a brush agent.
- An *environment* consists of a visualized two-dimensional brush model, a paper canvas, and an input boundary map.
- An *action* allows the controller to change the environment so as to influence the process of stroke generation.
- A *state* characterizes the environment. At each time step, the controller receives a state measurement and outputs an action. This causes a transition to a new state.
- A *reward* represents the quality of state transition. The reward provides the controller feedback on its immediate performance.

The controller receives a new state measurement, and the whole cycle is repeated until the task is accomplished. The intelligence of the agent, how to choose optimal actions, is the core issue in this procedure. The behavior of the brush agent is described by its *policy*, which outputs an action given a state. The goal of this optimal control problem is to find an optimal policy that maximizes the expected *return*, which is the expected cumulative reward over the course of interaction with the environment.

The process dynamics and the reward function, together with the sets of possible states and actions, constitute a Markov decision process. This is mathematically formalized below.

3.2 Markov Decision Process

Let us formulate the procedure of drawing a stroke as a Markov decision process (MDP).

An MDP consists of a tuple

$$(\mathcal{S}, \mathcal{A}, p_{\mathrm{I}}, p_{\mathrm{T}}, R), \tag{1}$$

where

- $\mathcal{S}$ is a set of continuous states,
- $\mathcal{A}$ is a set of continuous actions,
- p_{I} is the probability density of the initial state,
- $p_{\mathrm{T}}(\boldsymbol{s}'|\boldsymbol{s}, a)$ is the transition probability density from current state $\boldsymbol{s} \in \mathcal{S}$ to next state $\boldsymbol{s}' \in \mathcal{S}$ when taking action $a \in \mathcal{A}$,
- $R(\boldsymbol{s}, a, \boldsymbol{s}')$ is an immediate reward function for the transition from $\boldsymbol{s}$ to $\boldsymbol{s}'$ by taking action a.

Let $\pi(a|\boldsymbol{s};\boldsymbol{\theta})$ be a stochastic policy with parameter $\boldsymbol{\theta}$. This represents the conditional probability of taking action a given state $\boldsymbol{s}$. Let

$$h = (\boldsymbol{s}_1, a_1, \ldots, \boldsymbol{s}_T, a_T, \boldsymbol{s}_{T+1}) \tag{2}$$

be a trajectory of length T. The *return* (i.e., the discounted sum of future rewards) along h is defined as

$$R(h) = \sum_{t=1}^{T} \gamma^{t-1} R(\boldsymbol{s}_t, a_t, \boldsymbol{s}_{t+1}), \tag{3}$$

where $\gamma \in [0, 1)$ is the discount factor for the future reward. The expected return for parameter $\boldsymbol{\theta}$ is defined by

$$J(\boldsymbol{\theta}) = \int p(h|\boldsymbol{\theta}) R(h) \mathrm{d}h, \tag{4}$$

where

$$p(h|\boldsymbol{\theta}) = p(\boldsymbol{s}_1) \prod_{t=1}^{T} p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, a_t)\pi(a_t|\boldsymbol{s}_t, \boldsymbol{\theta}). \tag{5}$$

The goal of reinforcement learning (RL) is to find the optimal policy parameter $\boldsymbol{\theta}^*$ that maximizes the expected return $J(\boldsymbol{\theta})$:

$$\boldsymbol{\theta}^* \equiv \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\ J(\boldsymbol{\theta}). \tag{6}$$

3.3 Policy Gradient Method

We use a *policy gradient* algorithm [21] to solve the above RL problem. Here we briefly review the policy gradient method.

The policy parameter $\boldsymbol{\theta}$ is updated via gradient ascent as

$$\boldsymbol{\theta} \longleftarrow \boldsymbol{\theta} + \varepsilon \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}), \tag{7}$$

where ε is a learning rate. The gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ is given by

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \int \nabla_{\boldsymbol{\theta}} p(h|\boldsymbol{\theta}) R(h) \mathrm{d}h \\ &= \int p(h|\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log\, p(h|\boldsymbol{\theta}) R(h) \mathrm{d}h \\ &= \int p(h|\boldsymbol{\theta}) \sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log\, \pi(a_t|\boldsymbol{s_t}, \boldsymbol{\theta}) R(h) \mathrm{d}h,\end{aligned} \tag{8}$$

where we used the so-called *log trick*:

$$\nabla_{\boldsymbol{\theta}} p(h|\boldsymbol{\theta}) = p(h|\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log\, p(h|\boldsymbol{\theta}). \tag{9}$$

Since $p(h|\boldsymbol{\theta})$ is unknown, the expectation is approximated by the empirical average:

$$\nabla_{\boldsymbol{\theta}} \widehat{J}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log\, \pi(a_t^{(n)}|\boldsymbol{s}_t^{(n)}, \boldsymbol{\theta}) R(h^{(n)}), \tag{10}$$

where $\{h^{(n)}\}_{n=1}^{N}$ are N episodic samples with T steps and

$$h^{(n)} = (\boldsymbol{s}_1^{(n)}, a_1^{(n)}, \ldots, \boldsymbol{s}_T^{(n)}, a_T^{(n)}, \boldsymbol{s}_{T+1}^{(n)}). \tag{11}$$

Let us employ the Gaussian policy function with parameter $\boldsymbol{\theta} = (\boldsymbol{\mu}^\top, \sigma)^\top$, where $\boldsymbol{\mu}$ is the mean vector and σ is the standard deviation:

$$\pi(a|\boldsymbol{s};\boldsymbol{\theta}) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(a - \boldsymbol{\mu}^\top \boldsymbol{s})^2}{2\sigma^2}\right). \tag{12}$$

Then the derivatives of the expected return $J(\boldsymbol{\theta})$ with respect to the parameter $\boldsymbol{\theta}$ are given as

$$\nabla_{\boldsymbol{\mu}} \log \pi(a|\boldsymbol{s};\boldsymbol{\theta}) = \frac{a - \boldsymbol{\mu}^\top \boldsymbol{s}}{\sigma^2} \boldsymbol{s}, \tag{13}$$

$$\nabla_{\sigma} \log \pi(a|\boldsymbol{s};\boldsymbol{\theta}) = \frac{(a - \boldsymbol{\mu}^\top \boldsymbol{s})^2 - \sigma^2}{\sigma^3}. \tag{14}$$

Consequently, the policy gradients $\nabla_{\boldsymbol{\theta}} \widehat{J}(\boldsymbol{\theta})$ are given as

$$\nabla_{\boldsymbol{\mu}} J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N}(R(h^{(n)}) - b)\sum_{t=1}^{T}\frac{(a_t^{(n)} - \boldsymbol{\mu}^{\top}\boldsymbol{s}_t^{(n)})\boldsymbol{s}_t^{(n)}}{\sigma^2}, \tag{15}$$

$$\nabla_{\sigma} J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N}(R(h^{(n)}) - b)\sum_{t=1}^{T}\frac{\left(a_t^{(n)} - \boldsymbol{\mu}^{\top}\boldsymbol{s}_t^{(n)}\right)^2 - \sigma^2}{\sigma^3}, \tag{16}$$

where b is a *baseline* for reducing the variance of gradient estimates. The optimal baseline that minimizes the variance of the gradient estimate is given as follows [15]:

$$\begin{aligned} b^* &= \underset{b}{\operatorname{argmin}} \mathbf{Var}[\nabla_{\boldsymbol{\theta}} \widehat{J}(\boldsymbol{\theta})] \\ &\simeq \frac{\frac{1}{N}\sum_{n=1}^{N} R(h^{(n)}) \left\|\sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi(a_t^{(n)}|\boldsymbol{s}_t^{(n)};\boldsymbol{\theta})\right\|^2}{\frac{1}{N}\sum_{n=1}^{N} \left\|\sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi(a_t^{(n)}|\boldsymbol{s}_t^{(n)};\boldsymbol{\theta})\right\|^2}. \end{aligned} \tag{17}$$

Finally, the policy parameter $\boldsymbol{\theta} = (\boldsymbol{\mu}^{\top}, \sigma)^{\top}$ is updated as

$$\boldsymbol{\mu} \longleftarrow \boldsymbol{\mu} + \varepsilon \nabla_{\boldsymbol{\mu}} J(\boldsymbol{\theta}), \tag{18}$$

$$\sigma \longleftarrow \sigma + \varepsilon \nabla_{\sigma} J(\boldsymbol{\theta}). \tag{19}$$

4 Tailoring RL to Automatic Stroke Generation

In this section, we introduce Sumi-e brush styles and give a specific design of states, actions, and rewards tailored for our Sumi-e agent, as well as the design of the training session.

4.1 Brushing Styles

In oriental ink painting, there are several different brushing styles that characterize the paintings. In our system, we implement the *upright brush style* and the *oblique brush style* (see Figure 3).

In the upright brush style, the tip of the brush agent should be located on the medial axis of the shape of the expected stroke, and the bottom of the agent should be tangent to both sides of the boundary of the expected stroke shape. On the other hand, in the oblique brush stroke style, the tip of the agent should touch one side of the boundary; meanwhile, the bottom of the agent should be tangent to the other side of the boundary. In both styles, if the next footprint does not meet the above requirement, the next footprint will remain the same posture as the current one, but it merely makes transition to a new position by Action 1.

The choice of the upright brush style and the oblique brush style is exclusive and we ask a user to choose one of the styles in advance.

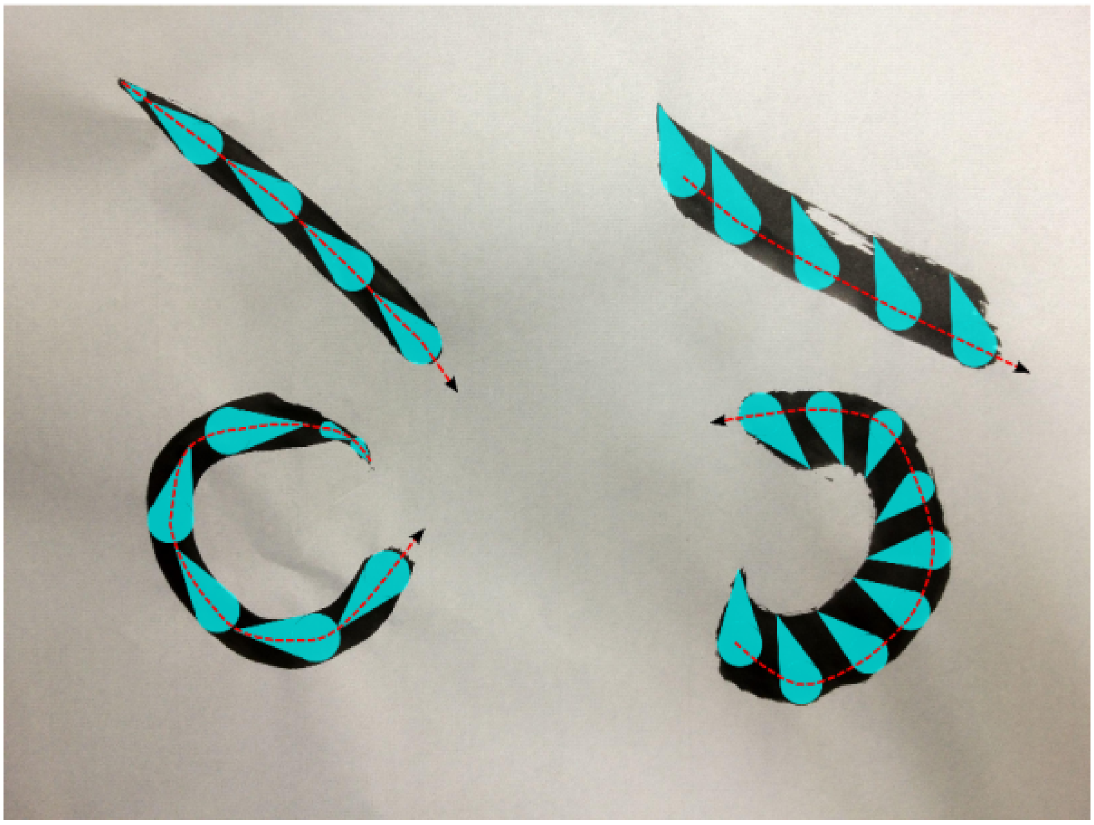


Figure 3: Brushing Styles. Left: Upright brush style. Right: Oblique brush style.

4.2 Design of States

As a state, we use the *global* measurement (the pose configuration of a footprint under the global Cartesian coordinate) and the *relative* measurement (the pose and the locomotion information of the brush agent relative to the surrounding environment). Here, our idea is to only use the relative measurement for calculating a reward and a policy, by which the agent can learn a drawing policy that is generalizable to new shapes. Below, we refer to the relative measurement as state $\boldsymbol{s}$; the global measurement is dealt with only implicitly.

Our relative state-space design consists of two parts: A current surrounding shape and an upcoming shape. More specifically, our state vector consists of the following six features:

$$\boldsymbol{s} = (\omega, \phi, d, \kappa_1, \kappa_2, l)^\top. \tag{20}$$

Each feature is defined as follows (see Figures 4):

- $\omega \in (-\pi, \pi]$: The angle of the velocity vector of the brush agent relative to the medial axis (see Figure 4(a)).
- $\phi \in (-\pi, \pi]$: The heading direction of the brush agent relative to the medial axis (see Figure 4(a)).

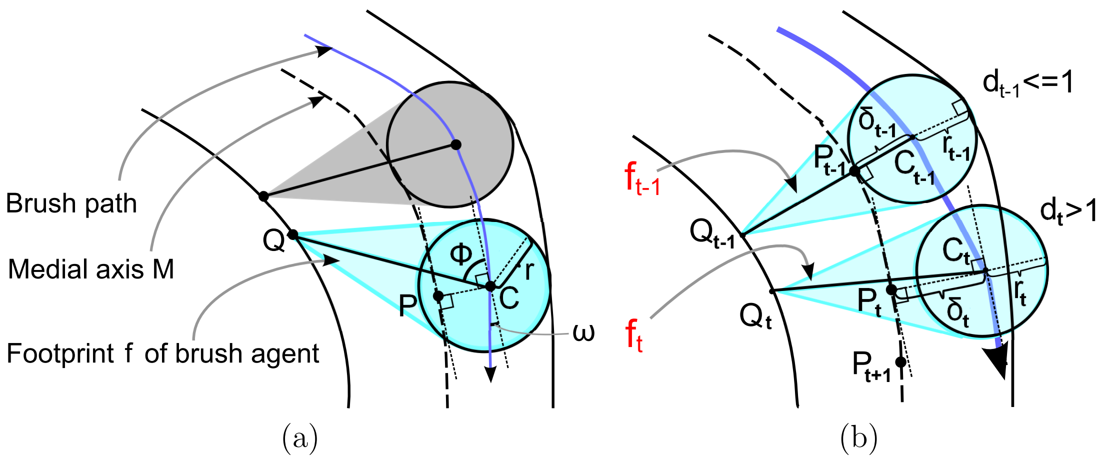


Figure 4: Illustration of the design of states: (a) Brush agent and its path. The brush agent consists of a tip Q and a circle with center C and radius r. (b) The ratio d of the offset distance δ over the radius r. Footprint f_{t-1} is inside the drawing area. The circle with center C_{t-1} and the tip Q_{t-1} touch the boundary on each side. In this case, $\delta_{t-1} \leq r_{t-1}$ and $d_{t-1} \in [0, 1]$. On the other hand, f_t goes over the boundary, and $\delta_t > r_t$ and $d_t > 1$. In our implementation, we restrict d to be in $[-2, 2]$. P is the nearest point on medial axis $\mathcal{M}$ to C.

- $d \in [-2, 2]$: The ratio of offset distance δ (see Figure 4(b)) from the center C of the brush agent to the nearest point P on the medial axis $\mathcal{M}$ over the radius r of the brush agent ($|d| = \delta/r$). d takes positive/negative values when the center of the brush agent is on the left-/right-hand side of the medial axis:

  - d takes the value 0 when the center of the brush agent is on the medial axis.
  - d takes a value in $[-1, 1]$ when the brush agent is inside the boundaries (for example, d_{t-1} in Figure 4(b))
  - The value of d is in $[-2, -1)$ or in $(1, 2]$ when the brush agent goes over the boundary of one side (for example, d_t in Figure 4(b)).

  In our system, the center of the agent is restricted within the shape. Therefore, the extreme value of d is ± 2 when the center of the agent is on the boundary.

- $\kappa_1, \kappa_2 \in (-1, 1)$: κ_1 provides the current surrounding information on the point P_t, whereas κ_2 provides the upcoming shape information on point P_{t+1}, as illustrated in Figure 4(b). The values are calculated as

$$\kappa_i = \frac{2}{\pi} \arctan\left(\frac{0.05}{\sqrt{r'_i}}\right), \tag{21}$$

  where r'_i is the radius of the curve. More specifically, the value takes 0/negative/positive when the shape is straight/left-curved/right-curved, and the larger its absolute value is, the tighter the curve is.

- $l \in \{0, 1\}$: A binary label that indicates whether the agent moves to a region covered by the previous footprints or not. $l = 0$ means that the agent moves to a region covered by the previous footprint. Otherwise, $l = 1$ means that it moves to an uncovered region.

4.3 Design of Actions

To generate elegant brush strokes, the brush agent should move inside given boundaries properly. Here, we define a 3-dimensional action to control the brush:

- Action 1: Movement of the brush on the canvas paper.
- Action 2: Scaling up/down of the footprint.
- Action 3: Rotation of the heading direction of the brush (see Figure 1(a)).

Since properly covering the whole desired region is the most important in terms of the visual quality, we regard the movement of the brush as the primary action (Action 1). More specifically, Action 1 takes a value in $(-\pi, -\pi]$ that indicates the offset turning angle of the motion direction *relative* to the medial axis of the shape of an expected stroke. In practical applications, the agent should be able to deal with arbitrary strokes in various scales. To achieve stable performance in different scales, we adaptively change the velocity as $r/3$, where r is the radius of the current footprint.

In our implementation, only Action 1 is determined by the Gaussian policy function in the MDP. The other actions (Actions 2 and 3) are automatically optimized to satisfy the assumption of the strokes. More specifically, depending on the style of the brush stroke, Actions 2 and 3 are chosen as follows (see also Section 4.1).

- Oblique brush stroke style: The tip of the agent is set to touch one side of the boundary, and the bottom of the agent is set to tangent with the other side of the boundary. If this is not possible by adjusting Actions 2 and 3, the new footprint will take the same posture as the previous one, but just transit to a new position by Action 1.
- Upright brush stroke style: The tip of the agent is chosen to travel along the medial axis of the shape.

This simplification allows us to reduce uncertainty in the calculation of the multi-dimensional actions using the policy function.

4.4 Design of Rewards

The reward function measures the quality of the brush agent's movement after taking an action at each time step. We design the reward to reflect the following two aspects:

- The distance between the center of the brush agent and the nearest point on the medial axis of the shape at the current time step. This detects whether the agent moves out of the region or travels backward from the correct direction.
- A change of the local configuration of the brush agent after executing an action. This detects whether the agent moves smoothly.

We formalize the above idea by defining the reward function as follows:

$$R(\boldsymbol{s}_t, a_t, \boldsymbol{s}_{t+1}) = \begin{cases} 0 & \text{if } f_t = f_{t+1} \text{ or } l_{t+1} = 0, \\ \dfrac{2 + |\kappa_1(t)| + |\kappa_2(t)|}{E^{(t)}_{\text{location}} + E^{(t)}_{\text{posture}}} & \text{otherwise,} \end{cases} \tag{22}$$

where f_t and f_{t+1} are footprints at time steps t and $t+1$, respectively. This reward design means that the immediate reward is zero when the brush is blocked by a boundary as $f_t = f_{t+1}$ or the brush is going backward to a region that has already been covered by previous footprints f_i $(i < t+1)$. $|\kappa_1(t)| + |\kappa_2(t)|$ adaptively increases immediate rewards depending on the difficulty of the current shape measured by the curvatures $\kappa_1(t)$ and $\kappa_2(t)$ of the medial axis.

$E^{(t)}_{\text{location}}$ measures the quality of the location of the brush agent with respect to the medial axis, defined by

$$E^{(t)}_{\text{location}} = \begin{cases} \tau_1\,|\omega_t| + \tau_2(|d_t| + 5) & d_t \in [-2, -1) \cup (1, 2], \\ \tau_1\,|\omega_t| + \tau_2\,|d_t| & d_t \in [-1, 1], \end{cases} \tag{23}$$

where d_t is the value of d at time t. τ_1 and τ_2 are weight parameters, which are chosen depending on the brush style. Our choice of τ_1 and τ_2 will be explained in Section 4.1. Since d_t contains information whether the agent goes over the boundary or not, as illustrated in Figure 4(b), the penalty $+5$ is added to E_{location} when the agent goes over the boundary of the shape.

$E^{(t)}_{\text{posture}}$ measures the quality of the posture of the brush agent based on neighboring footprints, defined by

$$E^{(t)}_{\text{posture}} = \Delta\omega_t/3 + \Delta\phi_t/3 + \Delta d_t/3, \tag{24}$$

where $\Delta\omega_t$, $\Delta\phi_t$, and Δd_t are changes in angles ω of the velocity vector, heading directions ϕ, and ratios d of the offset distance, respectively. The notation Δx_t denotes the normalized squared changes between x_{t-1} and x_t defined by

$$\Delta x_t = \begin{cases} 1 & \text{if } x_t = x_{t-1} = 0, \\ \dfrac{(x_t - x_{t-1})^2}{(|x_t| + |x_{t-1}|)^2} & \text{otherwise.} \end{cases} \tag{25}$$

For each style, we individually specify the reward function: $\tau_1 = \tau_2 = 0.5$ for the upright brush style and $\tau_1 = 0.1$ and $\tau_2 = 0.9$ for the oblique brush style.

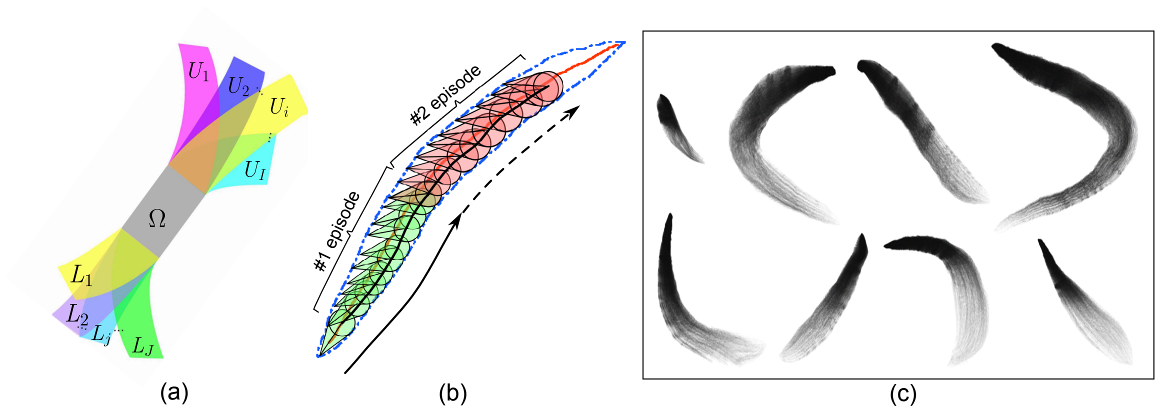


Figure 5: Policy training scheme. (a) Shape combination. Each shape ($U_i \cup \Omega \cup L_j, i = 1, 2, ..., I$ and $j = 1, 2, ..., J$) is combined with one of the upper regions U_i, the common region Ω, and the lower regions L_j. (b) Setup of policy training. (c) The brush library of single strokes in typical shapes. Only 8 out of 80 are shown here.

4.5 Design of Training Session

We propose to train the agent based on *partial* shapes, not the entire shapes. An advantage of using partial shapes is that various partial shapes can be generated from a single entire shape, which significantly increases the number and variation of training samples. Another merit is that the generalization ability to new shapes can be enhanced, because even when the entire profile of a new shape is quite different from that of training data, the new shape may contain similar partial shapes, as illustrated in Figure 5(a).

We provide a wide variety of partial shapes of strokes to the agent as training data. We invited Sumi-e experts to draw strokes in order to prepare an in-house stroke library. This library contains 80 digitized real single brush strokes that are commonly used in oriental ink painting. See Figure 5(c) for some examples. We extracted boundaries as the shape information and arranged them in a queue for training (see Figure 5(b)).

In the training scheme, the initial position of the first episode is chosen to be the start point $\mathcal{S}$ of the medial axis [3], and the direction to move is chosen to be the goal point $\mathcal{G}$, as illustrated in Figure 5(b). In the first episode, the initial footprint is set around the start point of the shape. In the following episodes, the initial footprint is set as either the last footprint in the previous episode or the footprint around the start point. It depends on whether the agent moved well or was blocked by the boundary in the previous episode.

For each policy, we repeat N episodes to collect data

$$H = [h^{(1)}, h^{(2)}, \dots, h^{(N)}], \tag{26}$$

where

$$h^{(n)} = [\boldsymbol{s}_1^{(n)}, a_1^{(n)}, \dots, \boldsymbol{s}_T^{(n)}, a_T^{(n)}, \boldsymbol{s}_{T+1}^{(n)}]. \tag{27}$$

We then use the data H to calculate the gradient of the return, $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$, and update the policy parameter M times to optimize the policy (see Section 3 again).

A full description of our policy training procedure is given in Algorithm 1.

Algorithm 1 Policy training.

```
Input $\mathcal{B}$: Set of training shapes
Input $R(s_{t-1}, s_t)$: Reward function
Input DB: Database of typical strokes shapes
Output $\pi$: Policy
Initialize policy $\pi_1$
for $m = 1$ to $M$ do
  // $M$ denotes the number of policy iterations
  for all $b \in \mathcal{B}$ do
    // Query the boundary, $\Omega_b$, to process.
    $\Omega_b = \text{doQuery}(b, \text{DB})$
    for $n = 1$ to $N$ do
      // $N$ denotes the number of repetitions
      Initialize the agent's state $\boldsymbol{s}_1$ and action $a_1$
      for $t = 1$ to $T$ do
        // $T$ denotes the length of a trajectory
        // Observe the agent's state $\boldsymbol{s}_{t+1}$.
        $\boldsymbol{s}_{t+1} = \text{exploreState}(\boldsymbol{s}_t, a_t, \Omega_b)$
        // Observe the agent's immediate reward $r_t$.
        $r_t = R(\boldsymbol{s}_t, \boldsymbol{s}_{t+1})$
        // Determine the agent's action $a_{t+1}$.
        $a_{t+1} = \pi_m(a|\boldsymbol{s}_t)$
        // Update the trajectory $h^{(n)}$.
        $h^{(n)} = h^{(n)} \bigcup (\boldsymbol{s}_t, a_t, \boldsymbol{s}_{t+1})$
      end for
    end for
    $H = (h^{(1)}, h^{(2)}, \ldots, h^{(N)})$
  end for
  // Update the policy.
  $\pi_{m+1} = \text{updatePolicy}(\pi_m, H)$
end for
$\pi = \pi_{M+1}$
```

Algorithm 2 Applying a learned policy to new shapes.

```
**Input** $\pi$: Learned policy
**Input** $\Omega$: Given boundary
**Input** $\mathcal{F}$: Brush style
**Input** $\mathcal{M}$: Medial axis of $\Omega$
**Input** $\mathcal{R}(x, y, l, \phi)$: Placement of the current footprint with the reference center point at $C(x, y)$, the scale length $l$ from $C$ to tip $Q$, and rotation over angle $\phi$.
**Output** $\psi$: Trajectory
Initialize start footprint $\mathcal{R}$, state $\boldsymbol{s}$, and end point $\mathcal{G}$
while dist($\mathcal{R}, \mathcal{G}$) $> \epsilon$ do
  // Select a new action by the optimal policy $\pi$.
  $a^{(1)} = \pi(a|\boldsymbol{s})$
  $(a^{(2)}, a^{(3)}) = \text{estimateActions}(\Omega, \mathcal{M}, \mathcal{F}, \mathcal{R}(x, y, l, \phi))$
  // Observe the agent state $\boldsymbol{s}'$.
  $\boldsymbol{s}' = \text{exploreNextState}(\boldsymbol{s}, a, \Omega)$
  // Observe the agent configuration $\mathcal{R}'$.
  $\mathcal{R}' = \text{getNextFootprint}(\mathcal{R}, a, \Omega)$
  if isInRegion($\mathcal{R}', \Omega$) then
    // Update the trajectory.
    $\psi = \psi \bigcup \mathcal{R}'$
    // Update the agent's current state $\boldsymbol{s}$.
    $\boldsymbol{s} = \boldsymbol{s}'$
    // Update the agent's current configuration $\mathcal{R}$.
    $\mathcal{R} = \mathcal{R}'$
  end if
end while
```

4.6 Applying Learned Policy to New Shapes

After learning a drawing policy, the brush agent applies the policy to covering given boundaries with strokes.

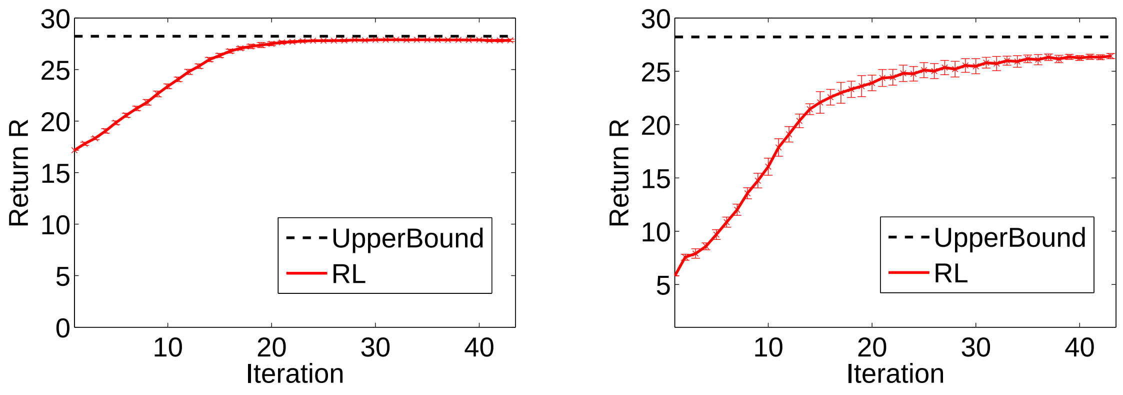
The location of the agent is initialized at the start point of a new shape. The agent then sequentially selects actions based on the learned policy and makes transitions until it reaches the goal point. See Algorithm 2 for details.

5 Experiments

In this section, we report experimental results.

We separately train policies for the upright brush style and the oblique brush style based on 80 single strokes in our library (see Figure 5(c)). The parameter of the initial policy is set as

$$\boldsymbol{\theta} = (\boldsymbol{\mu}^\top, \sigma)^\top = (0, 0, 0, 0, 0, 0, 2)^\top, \tag{28}$$



(a) Upright brush style (b) Oblique brush style

Figure 6: Average and standard deviation of returns obtained by the RL method over 10 trials and the upper limit of the return value.

where the first six elements correspond to the Gaussian mean and the last element is the Gaussian standard deviation. The agent collects $N = 300$ episodic samples with trajectory length $T = 32$. The discounted factor is set at $\gamma = 0.99$. The learning rate ε (see Section 3.3) is set at $0.1/\|\nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}}\|$.

In this experiment, to evaluate the performance of the trained policy, we investigate an obtained return which is a discounted cumulative aggregation of rewards along a trajectory starting from the initial starting point. We investigate the average return over 10 trials as functions of policy-update iterations. The return at each trial is computed over 300 training episodic samples. The average returns along the policy iteration are shown in Figure 6. The graphs show that the average returns sharply increase in an early stage and then they keep stable. We also plot the upper bounds of returns, which are the ideal maxima of the returns (i.e., receiving $r = 1$ for all steps).

In Table 1, we describe experimental results obtained by the DP method [22] with different numbers of footprint candidates in each step of the DP search. As the table shows, the execution time of the DP method increases significantly with the growth of the number of candidates. In the DP method, the best return value is 26.27 when the number of candidates is set to 180. This is comparable to the return obtained by the RL method (26.44). However, RL is around 50 times faster than the DP method. Figure 7 shows some exemplary examples of strokes generated by RL and DP. In the top two rows of Figure 7, the agent trained by RL is able to draw nice strokes with stable poses until the policy converges after the $30th$ policy-update iteration (see also Figure 6). However, in the DP method, the parameter of sampling discrete candidates at each step should be manually adjusted for each specific input shape. As illustrated in Figure 7, the results in first three columns (DP#5, #60 and #100) at the bottom two rows are not acceptable because neither the movement trajectories nor the poses of the neighbor footprints are stable.

We further apply our trained policy to more realistic shapes illustrated in Figure 8,

Table 1: Comparison of average returns and execution time between RL and DP for estimating a drawing trajectory on new shapes. Intel Core i7 2.70GHz is used for measuring the execution time.

Method	# Candidates	Return	Time [sec.]
DP	5	-0.60	$\mathbf{3.95 \times 10^1}$
	10	-0.10	1.01×10^2
	20	6.54	2.10×10^2
	30	12.17	3.25×10^2
	40	20.03	4.49×10^2
	50	20.66	5.73×10^2
	60	22.35	6.27×10^2
	70	22.33	7.48×10^2
	80	24.42	8.58×10^2
	90	25.48	9.74×10^2
	100	25.08	1.08×10^3
	110	25.80	1.19×10^3
	120	25.22	1.30×10^3
	130	25.43	1.40×10^3
	140	26.01	1.47×10^3
	150	24.50	1.68×10^3
	160	25.49	1.90×10^3
	170	25.89	2.03×10^3
	180	**26.27**	2.08×10^3
	190	26.04	2.30×10^3
	200	24.11	2.30×10^3
RL	$\varnothing$	**26.44**	$\mathbf{4.00 \times 10^1}$

which are not included in the training samples. We can observe that the policy obtained by the RL method can produce smooth and natural brush strokes in various unlearned shapes.

Finally, we employ our brush agent for photo conversion into an oriental ink painting style. This is carried out by manually specifying boundaries of single stroke segments on an original photo, as illustrated in the top-left graph of Figure 9, and feeding the obtained boundaries to the artist agent to generate strokes. The execution time for converting a photo to Sumi-e is proportional to the number of the strokes, and each single stroke can be independently processed within 1 minute. The conversion results are depicted in Figure 9, showing that the proposed approach is promising.

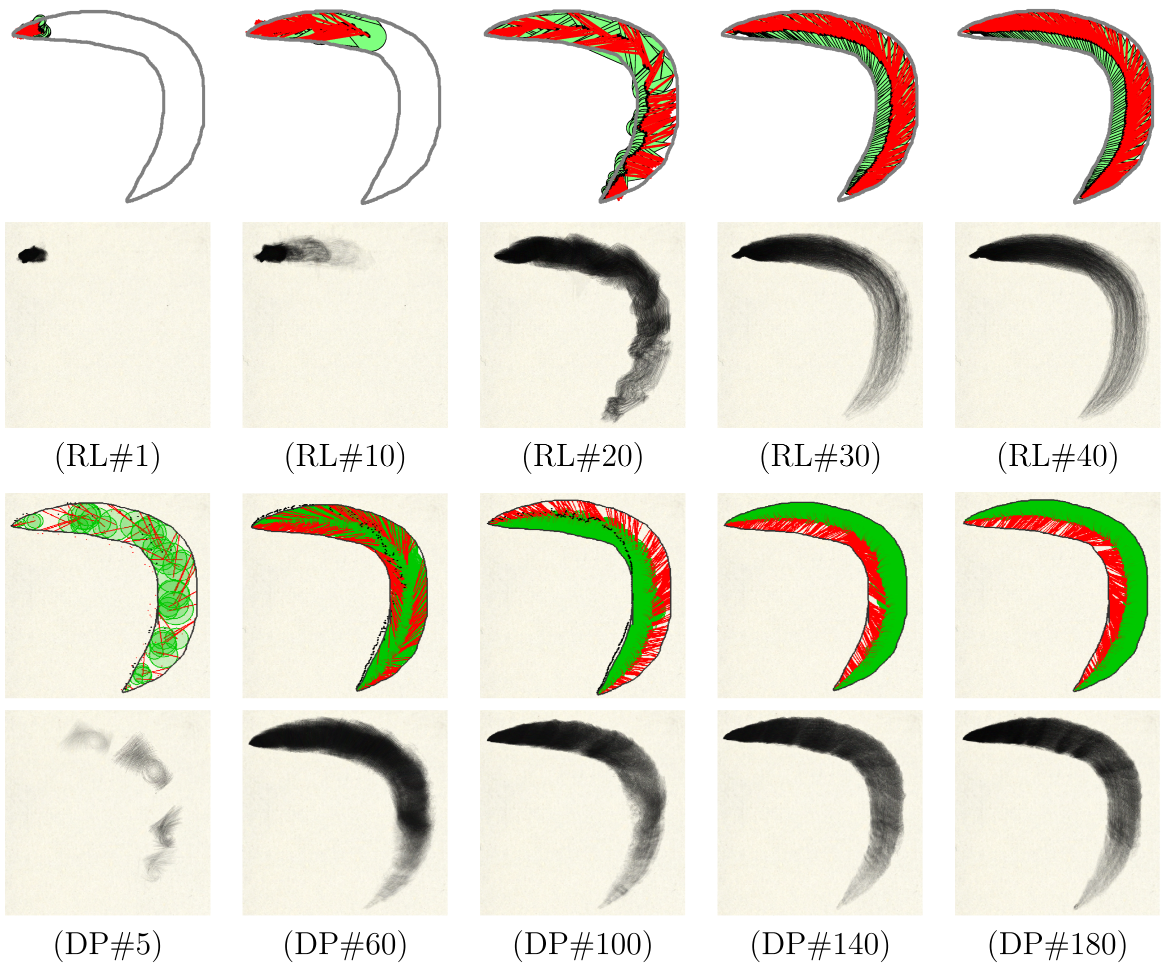


Figure 7: Examples of strokes generated by RL and DP. The top two rows show RL results over policy update iterations, while the bottom two rows show DP results for different numbers of footprint candidates. The red line denotes the link between the center and the tip of a footprint. The bottom circle of the footprint is marked in green.

6 Conclusions

In this paper, we applied reinforcement learning to oriental ink painting. This enabled automatic generation of smooth and natural strokes in arbitrary shapes.

Our contributions include careful designs of actions, states, immediate rewards, and training sessions. One of the key ideas was to design the state space of the brush agent to be *relative* to its surrounding shape. This allows a brush agent to learn a general drawing policy that is independent from a specific entire shape. This is a strong advantage over the existing dynamic programming approach. Another important idea of the proposed method was to train the brush agent with *partial* shapes. This contributed highly to enhancing the generalization ability to new shapes, because even when a new shape is quite different from training data as a whole, it often contains similar partial shapes.

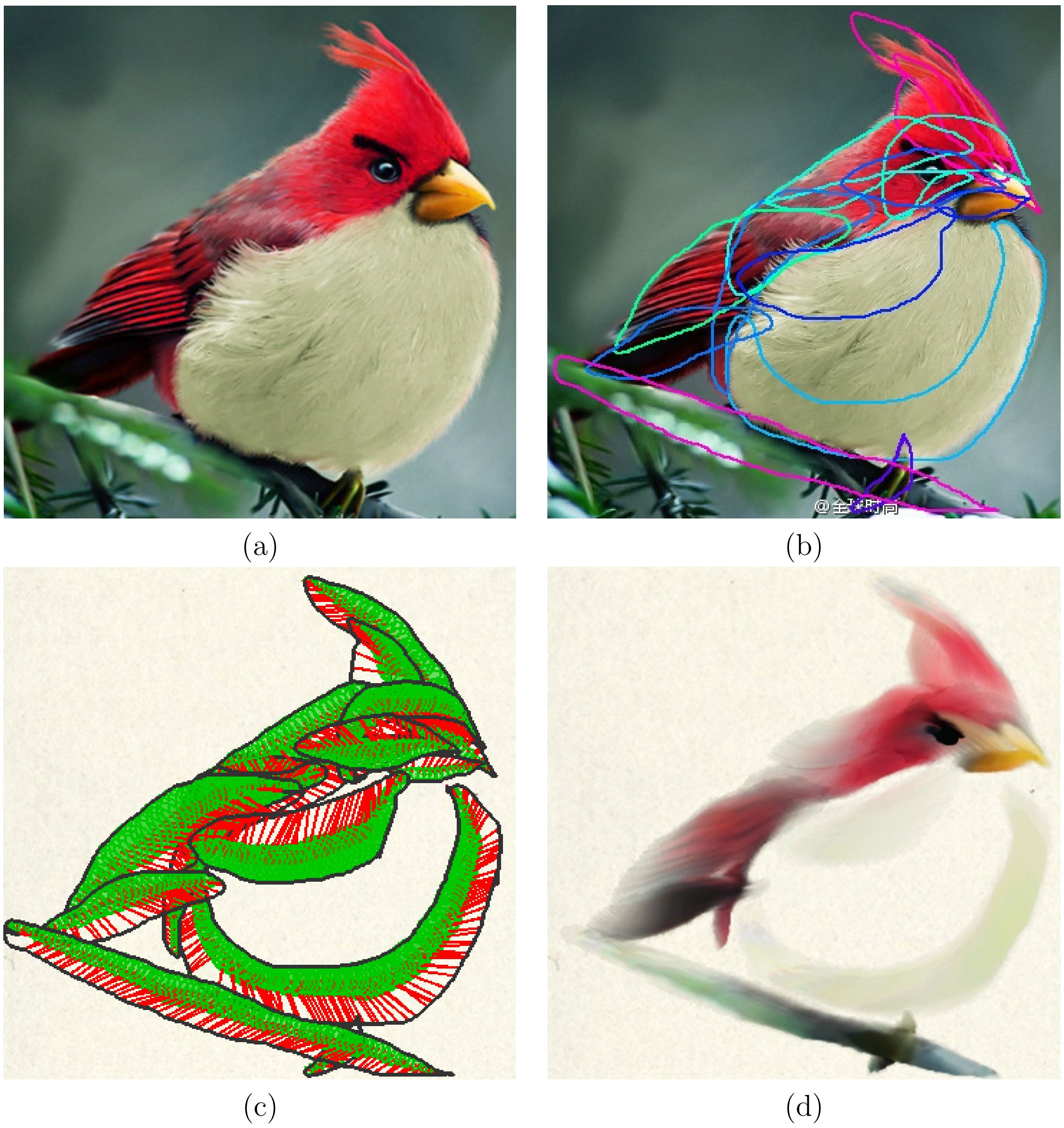


Figure 8: Results on new shapes. (a) Real photo. (b) User input boundaries. (c) Trajectories estimated by an RL-trained policy. (d) Rendering results for the RL-trained policy.

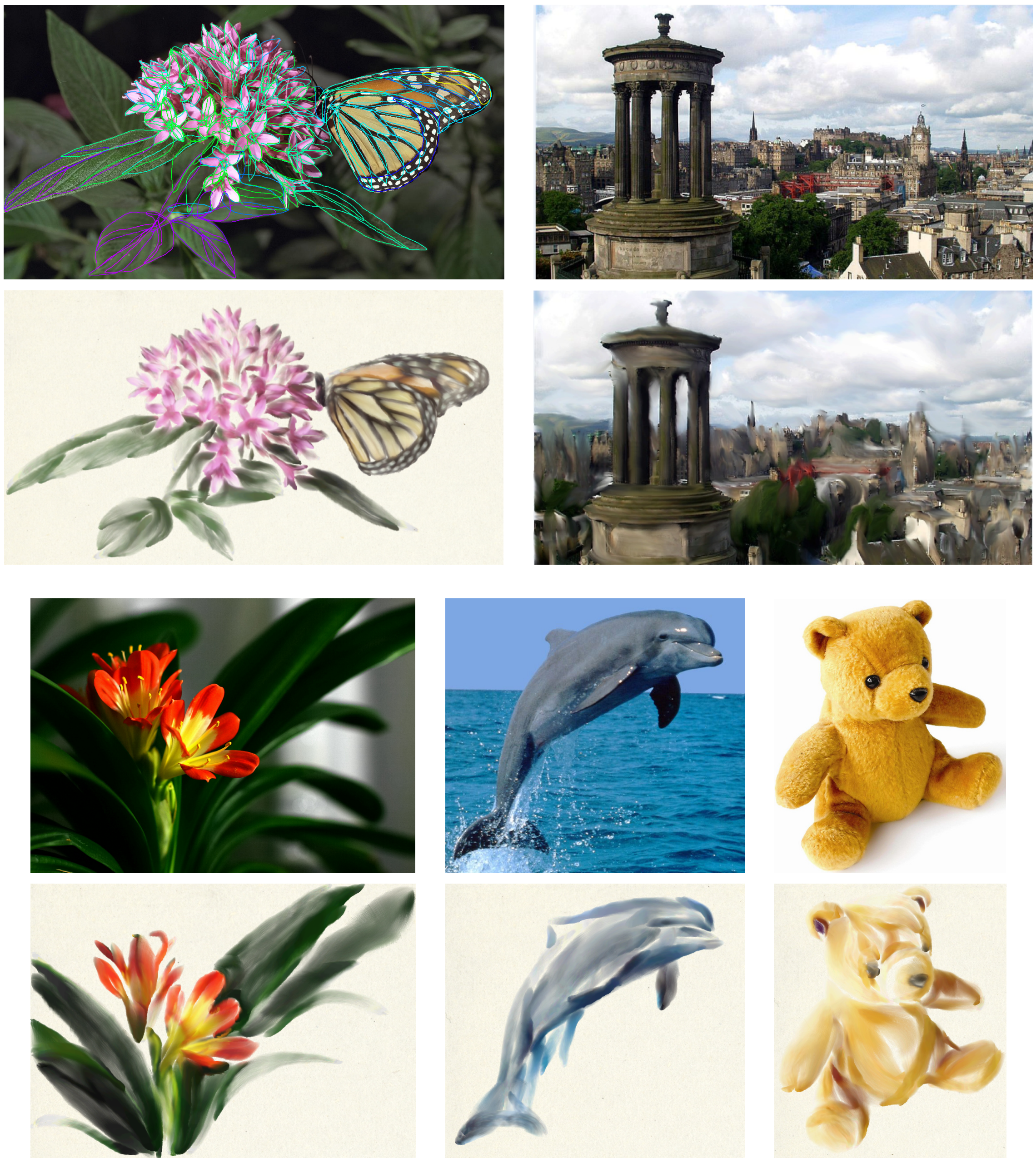


Figure 9: Results of photo conversion into an oriental ink style.

The experimental results demonstrated that our proposed method gives better performance than the dynamic programming approach with much less computation time. Furthermore, the agent trained by our reinforcement learning method can successfully draw unlearned new shapes with smooth and natural brush strokes. Also, applying photo conversion to an oriental style was demonstrated to be promising.

Our future work includes automatic design of reward functions. The use of *inverse reinforcement learning* [1] would be a promising approach for this purpose. In particular, such data-driven design of reward functions will allow us to automatically learn the style of a particular artist from his/her drawings.

Although the use of a simple Gaussian policy was demonstrated to work well in experiments, there is still room for further improvement in better designing a policy model. Another important future work is to automate the photo conversion procedure. For example the use of a *saliency map* [6] for extracting meaningful boundaries is a possible direction to be explored.

Acknowledgments

Ning Xie was supported by MEXT Scholarship 072231, Hirotaka Hachiya was supported by the FIRST program, and Masashi Sugiyama was supported by MEXT KAKENHI 23120004.

References

[1] P. Abbeel and A.Y. Ng, "Apprenticeship learning via inverse reinforcement learning," ICML, 2004.

[2] P. Asente, "Folding avoidance in skeletal strokes," SBM, pp.33–40, 2010.

[3] F. Chin, J. Snoeyink, and C.A. Wang, "Finding the medial axis of a simple polygon in linear time," Discrete Comput. Geom, pp.382–391, Springer-Verlag, 1995.

[4] N. Chu, W. Baxter, L.Y. Wei, and N. Govindaraju, "Detail-preserving paint modeling for 3D brushes," Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering, New York, NY, USA, pp.27–34, ACM, 2010.

[5] N. Chu and C.L. Tai, "Real-time painting with an expressive virtual Chinese brush," IEEE Computer Graphics and Applications, vol.24, no.5, pp.76–85, 2004.

[6] R. Desimone and J. Duncan, "Neural mechanisims of selective visual attention," Annual Review of Neuroscience, vol.18(1), pp.193–222, 1995.

[7] B. Gooch and A. Gooch, Non-Photorealistic Rendering, AK Peters Ltd, July 2001. ISBN: 1-56881-133-0.

[8] Q. Guo and T.L. Kunii, ""Nijimi" rendering algorithm for creating quality black ink paintings," Proceedings of Computer Graphics International, pp.152–159, July 2003.

[9] A. Hertzmann, "Painterly rendering with curved brush strokes of multiple sizes," Proceedings of SIGGRAPH, USA, pp.453–460, 1998.

[10] A. Hertzmann, "A survey of stroke based rendering," IEEE Computer Graphics and Applications, vol.23, pp.70–81, 2003.

[11] S.C. Hsu and H.H. Lee, "Drawing and animation using skeletal strokes," Proceedings of SIGGRAPH, USA, pp.109–118, ACM, 1994.

[12] G.D. Joshi and J. Sivaswamy, "A computational model for boundary detection," ICVGIP, pp.172–183, 2006.

[13] B.D.S. Lucian Busoniu, Robert Babuska and D. Ernst, Reinforcement Learning and Dynamic Programming Using Function Approximators, CRC Press, 2010.

[14] Y. Okabe, D. Maeda, S. Saito, and M. Nakajima, "An oriental rendering of lines by using discrete hmm," IEICE J90-D(1), pp.106–114, The Society for Imaging Science and Technology, 2007.

[15] J. Peters and S. Schaal, "Policy gradient methods for robotics," International Conference on Intelligent Robots and Systems, pp.2219–2225, 2006.

[16] S. Saito and M. Nakajima, "3D physics-based brush model for painting," Proceedings of SIGGRAPH, Conference Abstracts and Applications, USA, p.226, 1999.

[17] M. Shiraishi and Y. Yamaguchi, "An algorithm for automatic painterly rendering based on local source image approximation," Proceedings of the 1st International Symposium on Non-Photorealistic Animation and Rendering, USA, pp.53–58, ACM, 2000.

[18] S. Strassmann, "Hairy brushes," Proceedings of SIGGRAPH, USA, pp.225–232, ACM, 1986.

[19] M. Sugiyama, H. Hachiya, C. Towell, and S. Vijayakumar, "Geodesic Gaussian kernels for value function approximation.," Autonomous Robots, vol.25, no.3, pp.287–304, 2008.

[20] R.S. Sutton and A.G. Barto, Reinforcement Learning: An Introduction, The MIT Press, Cambridge, MA, 1998.

[21] R.J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," Machine Learning, vol.8, pp.229–256, 1992.

[22] N. Xie, H. Laga, S. Saito, and M. Nakajima, "Contour-driven Sumi-e rendering of real photos," Computers & Graphics, vol.35, no.1, pp.122–134, 2011.