

Im2Oil: Stroke-Based Oil Painting Rendering with Linearly Controllable Fineness Via Adaptive Sampling

Zhengyan Tong

418004@sjtu.edu.cn

Shanghai Jiao Tong University

Xiaohang Wang

xygz2014010003@sjtu.edu.cn

Shanghai Jiao Tong University

Shengchao Yuan

sc_yuan@sjtu.edu.cn

Shanghai Jiao Tong University

Xuanhong Chen

chen19910528@sjtu.edu.cn

Shanghai Jiao Tong University

Junjie Wang

dreamboy.gns@sjtu.edu.cn

Shanghai Jiao Tong University

Xiangzhong Fang

xzfang@sjtu.edu.cn

Shanghai Jiao Tong University

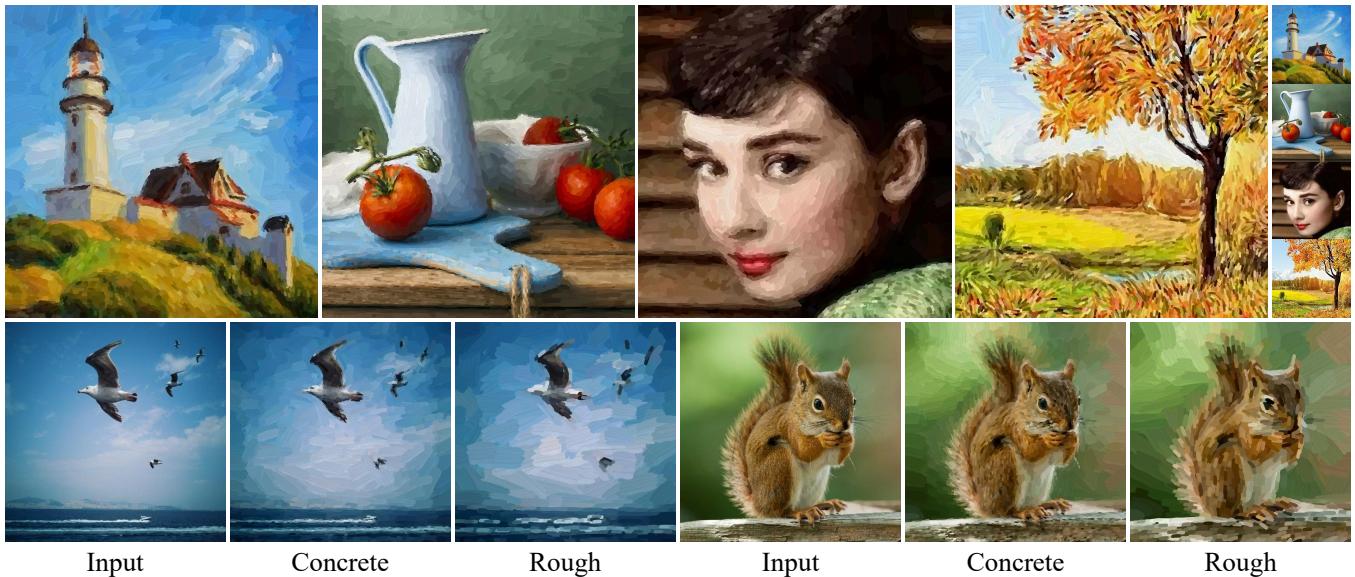


Figure 1: Oil paintings produced by our method. The first row shows four results with the inputs on the right. The second row shows two groups of oil paintings, each including a concrete result and a rough result. We recommend zooming in to see.

ABSTRACT

This paper proposes a novel stroke-based rendering (SBR) method that translates images into vivid oil paintings. Previous SBR techniques usually formulate the oil painting problem as pixel-wise approximation. Different from this technique route, we treat oil painting creation as an adaptive sampling problem. Firstly, we compute a probability density map based on the texture complexity of the input image. Then we use the Voronoi algorithm to sample a set of pixels as the stroke anchors. Next, we search and generate an individual oil stroke at each anchor. Finally, we place all the strokes on the canvas to obtain the oil painting. By adjusting the

hyper-parameter maximum sampling probability, we can control the oil painting fineness in a linear manner. Comparison with existing state-of-the-art oil painting techniques shows that our results have higher fidelity and more realistic textures. A user opinion test demonstrates that people behave more preference toward our oil paintings than the results of other methods. More interesting results and the code are in <https://github.com/TZYSJTU/Im2Oil>.

CCS CONCEPTS

- Applied computing → Fine arts.

KEYWORDS

stroke-based oil painting, adaptive sampling, fineness control

ACM Reference Format:

Zhengyan Tong, Xiaohang Wang, Shengchao Yuan, Xuanhong Chen, Junjie Wang, and Xiangzhong Fang. 2022. Im2Oil: Stroke-Based Oil Painting Rendering with Linearly Controllable Fineness Via Adaptive Sampling. In *Proceedings of the 30th ACM International Conference on Multimedia (MM '22)*, October 10–14, 2022, Lisboa, Portugal. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3503161.3547759>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

MM '22, October 10–14, 2022, Lisboa, Portugal.

© 2022 Association for Computing Machinery.

ACM ISBN 978-1-4503-9203-7/22/10...\$15.00

<https://doi.org/10.1145/3503161.3547759>

1 INTRODUCTION

Stroke Based Rendering (SBR) [19, 20] is both a classic topic and a challenging problem in the field of digital multimedia art. Different from the common image-to-image style transfers [10, 22, 59] that are usually based on pixel-wise mapping, SBR techniques create a painting by drawing discrete strokes on the canvas like real artists, which behaves highly intelligent and brings users impressive visual experience. As an imitation of the human creation process, SBR techniques usually follow some heuristic artistic principles, such as from coarse to fine [17, 23, 36] and edge enhancement [15, 35, 49]. To design an SBR algorithm (this paper focuses on the task of oil painting), researchers need to address three fundamental problems.

The first problem is how to search for strokes. Existing SBR techniques usually formulate oil painting creation as pixel-wise approximation, i.e., minimizing the pixel difference between the oil painting and the input image. The most common methods are greedy strategies, such as [17] drew new strokes at the largest error point. Some other methods [13, 18] randomly perturbed the strokes to reduce the sum-of-squares difference. Recent research [23] uses deep reinforcement learning to train an oil painting agent by optimizing the L_2 distance, while [36, 60] train neural networks to predict strokes by optimizing the L_1 loss. In this paper, we rethink the oil painting problem and come up with a highly interpretable assumption: areas with complex textures should be painted with small and dense strokes to maintain high fidelity; while areas with smooth textures could be painted with large and sparse strokes since it will not cause obvious visual degradation. Based on this idea, we formulate oil painting creation as an adaptive sampling problem. Firstly, we translate the input image into a probability density map whose probability values are determined by its local texture complexity. Pixels with complex textures have relatively large probabilities, while pixels with smooth textures have relatively small probabilities. Then we use the Voronoi algorithm to sample a set of pixels from the probability density map as the stroke anchors, whose number is proportional to the input image's global texture complexity. At each anchor, we generate an individual stroke based on its nearby texture features.

The second problem is how to render strokes. We divide existing SBR methods into model rendering and template rendering. Model rendering means the stroke is rendered by some handcrafted math models such as the Bézier curve in [23] and the B-spline in [20]. Template rendering means the stroke is derived from a template image, such as the brush dictionary in [57] and the primitive brush in [36]. In this paper, we adopt the method of template rendering to produce photo-realistic strokes: we use a gray image of a real oil brush as the stroke template and transform it into various strokes according to some attribute parameters.

The third problem is how to control the oil painting's fineness. People usually enjoy fine-grained results, while some others may prefer abstract results. Therefore, SBR techniques usually have a fineness control mechanism: [17] uses a threshold to limit the maximum pixel error of the painting; [15] divides the image into layers of different frequencies; [23, 36, 60] specify the stroke number in each block manually. However, these mechanisms are not user-friendly because they cannot control the fineness in a linear manner. For example, it is hard to predict how much the fineness will improve

when reducing the error threshold by 20% or using 20% more strokes. Obviously, a linear fineness control mechanism will make it easier for users to tune hyper-parameters and get the desired fineness. In this paper, the fineness of our oil painting is controlled by a hyper-parameter, maximum sampling probability (p_{max}), which not only determines the stroke number but also limits the stroke's maximum and minimum size. By adjusting this hyper-parameter, we can produce oil paintings with linearly changed fineness from concrete to rough. In Figure 1, the first row shows four concrete results using $p_{max} = \frac{1}{4}$, whose inputs are the small images on the right. The second row shows two groups of oil paintings each including a concrete ($p_{max} = \frac{1}{4}$) result and a rough ($p_{max} = \frac{1}{16}$) result. More discussion about fineness control is in Section 4.2.

Having addressed all these three problems, we presented this paper and titled it "Im2Oil" for short, which means translating an image into oil paintings. To prove the application value of our method, we compare it with three state-of-the-art SBR oil painting techniques [23, 36, 60]. The qualitative comparison shows that our method could generate oil paintings with more realistic textures, higher fidelity, and fewer artifacts. For quantitative comparison, we designed a Mean Opinion Score (MOS) test and invited 115 volunteers to rate the oil paintings produced by the above methods. The MOS test shows there exist obvious gaps among these methods, and the results of our method got the highest user rating.

Our contribution can be summarized as follows:

- We rethink the oil painting problem from the perspective of adaptive stroke anchor sampling rather than the commonly followed pixel-wise approximation technique route.
- We can linearly control the fineness of our oil painting by varying the hyper-parameter maximum sampling probability (p_{max}) to produce arbitrarily concrete or rough results.
- Both the qualitative comparison and the user opinion test demonstrate that our method is superior to state-of-the-art SBR oil painting techniques.

2 RELATED WORK

2.1 Stroke-Based Rendering

Stroke-Based Rendering (SBR) is a classic artificial intelligence topic that teaches machines to paint like artists. SBR tasks mainly include stippling [4, 21, 28, 30, 39, 40, 43, 47, 52], oil painting [11, 13, 17, 18, 23, 27, 31, 33, 35, 36, 57, 60], tile mosaics [5–8, 14, 27, 29, 48], pen-and-ink drawing [27, 44–46, 53, 54], pencil sketching [9, 12, 49], watercolor painting [3, 27, 51, 55, 56], and vector-field visualization [16, 24, 32, 37, 42, 50]. Technically, SBR algorithms include greedy strategies [11, 17, 24, 35], optimization or relaxation [4, 14, 28, 47], and trial-and-error methods [13, 18, 50]. In recent years, researchers have proposed many modern SBR techniques including RL watercolor painting [55, 56], RNN-based image doodling [12], CNN-based character writing [58], DRL oil painting [23], Paint Transformer [36], and SBR style transfer [31, 60].

2.2 Voronoi Related SBR Techniques

In this paper, we use the Voronoi algorithm to sample pixels as the stroke anchors. The Voronoi diagram [1] was first applied to the SBR task stippling in [4] by Deussen *et al.* who use evenly spaced



Figure 2: Overall Framework. Firstly, we use the input image to sample a total of K pixels as anchors. Next, we search and generate an individual oil stroke at each anchor, indicated as $\{S_1, S_2, S_3, \dots, S_K\}$. These strokes are placed one by one on the canvas during the rendering process. Finally, all the strokes compose the oil painting result.

points with different radii to approximate gray images. Inspired by this, Adrian Secord proposed weighted Voronoi stippling [47] by varying the point distribution rather than the point radius. For generalization, Hiller *et al.* [21] use various small objects with orientation as basic elements of Voronoi stippling. Kim *et al.* [28] combined the Voronoi diagram and the Jump Flooding Algorithm to create feature-guided stippling. Ma *et al.* [40] use incremental Voronoi sets for instant stippling. Lindemeier *et al.* [34] place new strokes between existing ones by maximizing the minimal distance in the Voronoi diagram. Besides, the Voronoi diagram could also be applied to simulating tile mosaics [6, 7, 14].

3 METHOD

The overall framework of our method is shown in Figure 2. Firstly, we use the *Input* image to sample a total of K pixels as *Anchors*. Next, we search and generate an individual *Oil Stroke* at each anchor, indicated as $\{S_1, S_2, S_3, \dots, S_K\}$. The obtained strokes are placed one by one on the canvas during the *Rendering Process*. Eventually, all the strokes compose the final result *Oil Painting*. In the following, we will introduce our method in detail.

3.1 Adaptive Anchor Sampling

As introduced in Section 1, we vary the stroke size and density in areas of different texture complexity. Since each stroke corresponds to an individual anchor, controlling the distribution of the strokes is controlling the distribution of their anchors. This section shows how to obtain the stroke anchors by adaptive sampling. Firstly, we use the input image I to calculate a probability density map I_p . Pixels in I_p with larger values have a larger probability of being sampled as an anchor. Generally, areas with complex textures (high frequency) are often accompanied by significant gradient changes, while areas with smooth textures (low frequency) usually have small gradients. However, the gradient map is not suitable to directly use as the probability density map because large gradients may only

exist on the narrow edges. To paint fine-grained contours, it is not enough to enhance only the edges, but also the neighboring areas of the edges. In other words, we should improve the sampling probability of the pixels around the edges. This can be solved by applying a smoothing operation on the gradient map so that the large gradients will diffuse to their neighboring areas. Specifically, we use the Sobel filter to extract gradients and use the Mean filter to smooth the gradient map. Both filters use a 5×5 window. Then the smoothed gradient map is normalized to an interval $[p_{min}, p_{max}]$ as the probability density map I_p , where $0 < p_{min} < p_{max} \leqslant 1$. Here p_{max} is the maximum sampling probability, and p_{min} is the minimum probability. $p_{min} > 0$ ensures that the pixels in the most smooth area could be sampled as anchors, so that some strokes will be drawn to this area. $p_{max} \leqslant 1$ means there can be at most one anchor at each pixel. In practice, we use $p_{min} = \frac{1}{100}p_{max}$, thus p_{max} is the only hyper-parameter that need to be specified by users. Now add up all the pixel values (sampling probability) in I_p , we use the sum K as the number of stroke anchors. Therefore, our method could automatically determine the stroke number based on the global texture complexity of the input image and the hyper-parameter maximum sampling probability p_{max} .

Algorithm 1 Rejection Sampling

Given: probability density map I_p , anchor number K ;
Output: K anchors;

- 1: Randomly generate a probability density map I_u which has the same shape as I_p , and $I_u \sim \text{Uniform}(0,1)$;
 - 2: Randomly select a pixel (x, y) . If $I_u(x, y) \leq I_p(x, y)$, sample this pixel; else, reject this pixel.
 - 3: Repeat Step 2 until sampling K different pixels.
-

Next, we use Rejection Sampling [41] to sample K pixels from the probability density map I_p as the stroke anchors, as shown in Algorithm 1: we use a uniform distribution $I_u \sim \text{Uniform}(0,1)$ to

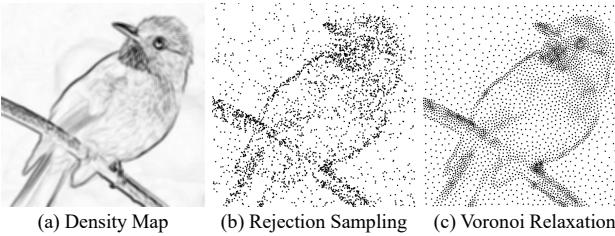


Figure 3: (a) is the probability density map. (b) is the rejection sampling result. We use $p_{max} = \frac{1}{4}$ and totally sample $K = 3068$ anchors. (c) is the Voronoi relaxation result.

sample the probability density map I_p ; every time we randomly select a pixel (x, y) , if $I_u(x, y) \leq I_p(x, y)$, then this pixel is sampled (else, rejected); repeat this operation until K different pixels are sampled. By this means, pixels in I_p with larger probabilities are more likely to be sampled. In Figure 3, (a) is the visualization of a probability density map (the input is in Figure 5). (b) is the rejection sampling result of (a). We use $p_{max} = \frac{1}{4}$, and $K = 3068$. The sampled anchors look messy because rejection sampling is just a rough approximation to the probability density map. Ideally, the distribution of the anchors should be as close as possible to the probability density map. This can be solved by applying the Voronoi algorithm (Lloyd's method [38]) to optimize (relax) the position of the anchors. Lloyd's method is also known as the famous K-means clustering [2]. As shown in Algorithm 2: the K anchors obtained by rejection sampling are used as the initial K centroids for pixel clustering; the probability density map I_p is used as the weight map of the pixels; each pixel in I_p is divided into the cluster of its nearest (coordinate distance) centroid; the new centroids of these clusters are calculated using the weight map I_p ; repeat the above clustering process for N iterations to optimize the centroid coordinates, which are the final positions of the anchors. In this way, the distribution of the anchors will gradually tend to the probability density map. As shown in Figure 3, (c) is the Voronoi relaxation result of (b) using Algorithm 2 for $N=15$ iterations. The distribution of the anchors in (c) looks much closer to (a) than in (b). In regions with smooth textures (such as the background), the anchors distribute sparsely and evenly, which mainly depends on the minimum sampling probability p_{min} . In regions with significant texture changes (such as the contours), the anchors gather densely while maintaining a certain distance, which mainly depends on the maximum sampling probability p_{max} . Actually, for any pixel, assume its sampling probability is p , there will be one anchor in its neighboring $\frac{1}{p}$ pixels (approximately). While for any anchor, assume its sampling probability is p , there will be no other anchors in its neighboring $\frac{1}{p}$ pixels (approximately). In practice, we fix $N = 15$, because after 15 iterations, the anchor distribution will have been close enough to the probability density map.

3.2 Stroke Searching and Generation

This section introduces how to search and generate a stroke at each anchor. Our stroke model is shown in Figure 4 (a) with the shape of rectangle. The red point indicates the anchor. The rotation angle

Algorithm 2 Voronoi Algorithm

Given: initial K centroids, weight map I_p , iterations N ;

Output: optimized K centroids;

- 1: For each pixel in I_p , calculate its coordinate distance to each centroid;
 - 2: Divide each pixel into the cluster of its nearest centroid;
 - 3: Use the weight map I_p to calculate the new centroid coordinates of these K clusters;
 - 4: Repeat step 1,2,3 for N iterations.
-

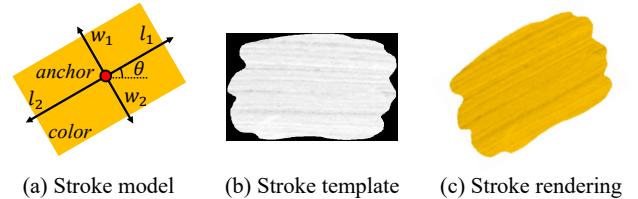


Figure 4: Stroke illustration. (a) is the stroke parameter model. (b) is the stroke template image. (c) is the stroke rendering result using the parameters in (a) and the template in (b).

Algorithm 3 Length Searching

Input: anchor coordinate (x_0, y_0) , searching direction α , input image (H, S, V) , H channel threshold t_H , V channel threshold t_V , step length δ ;

Output: length L ;

- 1: $(x, y) \leftarrow (x_0, y_0)$, $L \leftarrow 0$;
 - 2: **do**
 - 3: $L \leftarrow L + \delta$;
 - 4: $(x, y) \leftarrow (x, y) + (\delta \cdot \cos \alpha, \delta \cdot \sin \alpha)$;
 - 5: **while** $|H(x, y) - H(x_0, y_0)| < t_H$ and $|V(x, y) - V(x_0, y_0)| < t_V$
 - 6: **return** L
-

θ is the direction of the stroke. l_1 is the length extending along the direction of θ ; l_2 is the length extending along the direction of $(\theta + \pi)$; w_1 is the width extending along the direction of $(\theta + \frac{\pi}{2})$; w_2 is the width extending along the direction of $(\theta - \frac{\pi}{2})$. Besides, we use the anchor pixel's color as the stroke color.

Firstly, we introduce how to determine the stroke direction θ . When human artists paint, they usually draw strokes along the object's edges. That is because drawing strokes across the edges will result in a rough contour [35] while drawing strokes along the edges helps to produce accurate shapes [15]. Therefore, the stroke direction θ should follow the tangent of the edges. Specifically, we use the ETF vector field (Edge Tangent Flow [25, 26]) to find the tangent of the edges and use the anchor's ETF direction as θ . Here briefly reviews the calculation of ETF: compute the input image's gradient vector field (modulus and direction at each pixel); rotate the directions by $\frac{\pi}{2}$; tend the direction of the pixels with a small modulus to the direction of its nearby pixels with a large modulus, while the modulus is not changed. In this way, the ETF direction will approximately follow the tangent of its nearby edges. That is why it calls "Edge Tangent Flow".

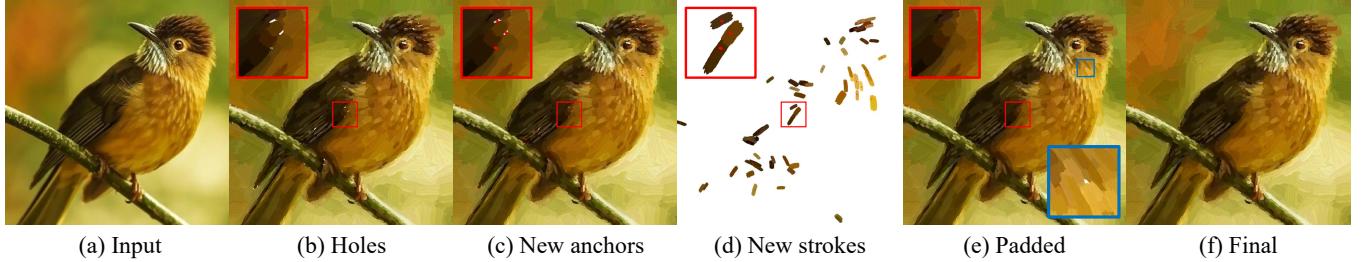


Figure 5: Illustration of the hole padding operation, please zoom in to see. (a) is the input. (b) is the result before padding. In (c), the small red dots are the new anchors. (d) shows the new strokes. (e) is the result of using (d) to pad (b), the zoomed in area with a blue border shows there still exists an uncolored hole. (f) is the final result without any uncolored pixels.

Next, we introduce how to determine the stroke length l_1, l_2 and width w_1, w_2 , which is described in Algorithm 3. We use the anchor coordinate (x_0, y_0) as the start point for searching. The searching direction $\alpha = \theta, \theta + \pi, \theta + \frac{\pi}{2}, \theta - \frac{\pi}{2}$ for l_1, l_2, w_1, w_2 , respectively. The input image is converted to the HSV color space since HSV (Hue, Saturation, Value) have more intuitive meanings than RGB. In every step, the point (x, y) moves length δ along the searching direction α (we fix $\delta = 1$ pixel). During searching, two conditions constrain the difference between the point (x, y) and the anchor (x_0, y_0) . $|H(x, y) - H(x_0, y_0)| < t_H$ restricts the stroke from being drawn to regions with much different hue. $|V(x, y) - V(x_0, y_0)| < t_V$ restricts the stroke from being drawn to regions with much different brightness. In practice, we fix $t_H = \frac{\pi}{3}$ and $t_V = 15$ as the thresholds. We do not set a condition for the S channel because the visual difference caused by saturation is not as obvious as hue and brightness. Besides, we use L_{min} and L_{max} as the minimum and maximum length to clip the searching length L . L_{min} and L_{max} are automatically determined by the anchor's sampling probability. As described at the end of Section 3.1, assume the anchor's sampling probability is p , there will be no other anchors in its neighboring $\frac{1}{p}$ pixels (approximately), whose area equals to a $p^{-\frac{1}{2}} \times p^{-\frac{1}{2}}$ square window. In view of this, it is reasonable to make L_{max} proportional to $p^{-\frac{1}{2}}$. In practice, we use $L_{max} = p^{-\frac{1}{2}}$ for w_1 and w_2 , and we use $L_{max} = 3 \times p^{-\frac{1}{2}}$ for l_1 and l_2 . That's because the stroke length are generally larger than width. As for L_{min} , we use $L_{min} = p_{max}^{-\frac{1}{2}}$ for w_1 and w_2 and $L_{min} = 3 \times p_{max}^{-\frac{1}{2}}$ for l_1 and l_2 . L_{min} limits the fidelity (detail resolution) that the oil painting can achieve, which essentially depends on the maximum sampling probability p_{max} .

As introduced in Section 1, we use the method of template to render strokes. Figure 4 (b) shows our oil stroke template, which is a gray image composed of a background mask and a foreground texture. Only the foreground is the actually used part. The mask with such an irregular shape looks much more natural than a rigid rectangle. The texture follows the long side of the stroke to reflect the drawing direction. Now given the stroke parameters $\theta, l_1, l_2, w_1, w_2$, and HSV color (h, s, v) , the stroke template is resized to length $l_1 + l_2$, width $w_1 + w_2$, and is rotated by angle θ . Then the template is colored by the following method. Take constant h and s as the stroke's H channel and S channel. For the V channel, the value varies with the texture of the template. Use T to denote the template image and use g_m to denote its grayscale average. The

template T is multiplied by $\frac{v}{g_m}$ as the V channel of the stroke. In this way, the mean energy of the V channel will not change because $Mean(\frac{v}{g_m} T) = \frac{v}{g_m} Mean(T) = v$. As shown in Figure 4, (c) is the stroke rendering result using the parameters in (a) and the template in (b). In addition to the stroke template in Figure 4 (b), we also offer a set of alternative templates with different textures, which is discussed in the [supplementary material](#).

3.3 Painting Process

By the methods in Section 3.2, we could search and generate a stroke at each anchor. Now we place the strokes one by one on the canvas to produce the oil painting. During this process, some strokes will overlap with each other. In this case, we make the newly drawn stroke completely cover the previous ones, which is consistent with the real oil painting. Therefore, different stroke drawing orders will lead to different final results. If the stroke number is K , then we have $K!$ different orders, and there must be an optimal solution that minimizes the pixel difference between the painting result and the original image. However, the factorial of K is too large to enumerate each order and find the optimal solution. Fortunately, we can quickly find an approximate solution. Review the anchor sampling method in Section 3.1 and the stroke length clipping mechanism in Section 3.2. Areas with complex textures are sampled with dense anchors and drawn with small strokes, while areas with smooth textures are sampled with sparse anchors and drawn with large strokes. Obviously, large strokes are more likely to cause errors, while small strokes are usually more accurate. In view of this, we sort the strokes by their area size and paint them from the largest to the smallest (greedy strategy).

As shown in Figure 5, (a) is the input image and (b) is the oil painting result after adding all the strokes to the canvas. Zoom in (b), it can be seen that a few small white holes are not colored. The enlarged area with a red border shows some of the holes. One reason for this phenomenon is that the mask of the stroke template discards some border regions and these regions happen to be not colored by any other strokes during the painting process. Besides, the noise in the image may influence stroke searching and cause some holes. Whatever, the current algorithm cannot guarantee to paint all the pixels. We solve this problem by the following padding method. Firstly, we find the connected domain of each uncolored hole and calculate the centroid of the connected domain. As shown in Figure 5 (c), the tiny red points are the centroids of the uncolored

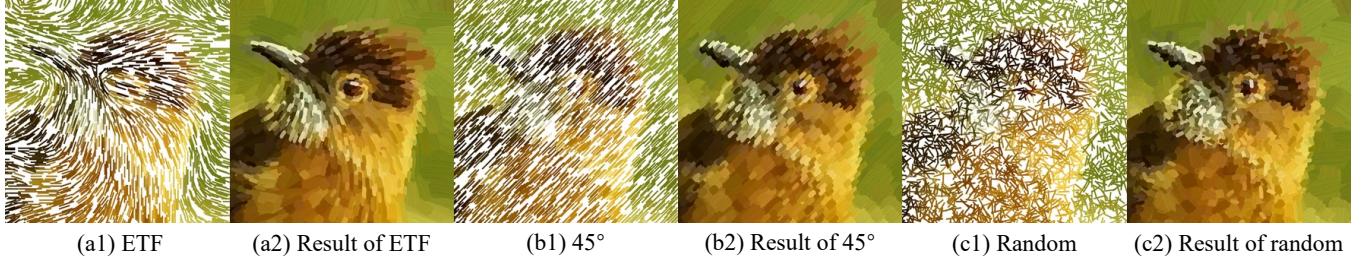


Figure 6: Comparison of different stroke direction. (a1) is the illustration of the ETF vector field; (b1) is the illustration of a constant vector field fixed at 45° ; (c1) is the illustration of a random vector field; (a2)(b2)(c2) are the corresponding oil painting results whose strokes are guided by the direction of (a1)(b1)(c1), respectively.

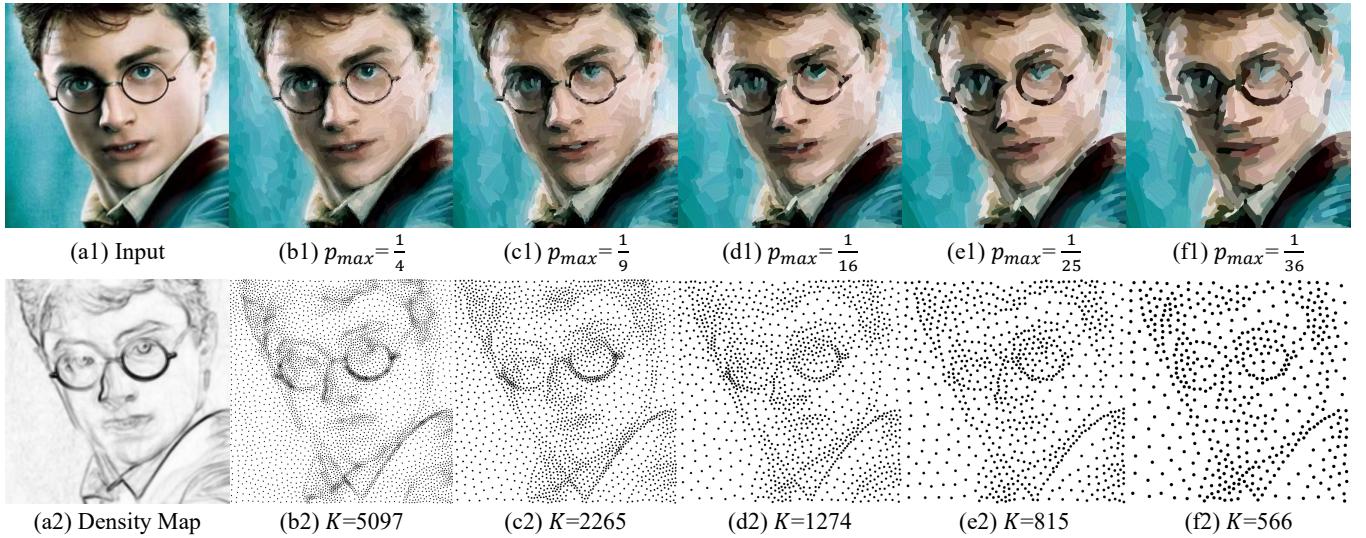


Figure 7: Illustration of oil painting fineness control. (a1) is the input; (a2) is the probability density map; (b1)(c1)(d1)(e1)(f1) are the oil painting result using $p_{max} = \{\frac{1}{4}, \frac{1}{9}, \frac{1}{16}, \frac{1}{25}, \frac{1}{36}\}$, respectively; (b2)(c2)(d2)(e2)(f2) are the corresponding anchors of (b1)(c1)(d1)(e1)(f1), which samples $K = \{5097, 2265, 1274, 815, 566\}$ anchors, respectively.

holes. Next, we use these centroids as new anchors to search and render strokes on a white canvas, obtaining (d). The tiny red points in (d) are the new anchors, whose positions are the same as those in (c). Now use (d) to pad (b), which means the value of those uncolored pixels in (b) are taken from the corresponding pixels in (d), while the value of those already colored pixels in (b) remain unchanged. The padding result is shown in (e). Although most of the holes disappear, the padding operation may not eliminate all the uncolored pixels at one time. The enlarged area with a blue border in (e) shows that one hole still exists. Therefore, the padding operation will usually iterate several times (empirically, three times in most cases) before eliminating all the holes. Finally, we get the oil painting result with no uncolored pixels, as shown in (f).

4 ABLATION STUDY

4.1 Stroke Direction

As described in Section 3.2, we use the Edge Tangent Flow vector field [25, 26] as the stroke direction guidance. The ablation study

of ETF is shown in Figure 6: (a1) is the visualization of the ETF direction, where the short lines denote the local ETF direction; (a2) is the oil painting result using ETF as its stroke direction guidance; (b1) is the visualization of a constant vector field with the direction of 45° ; (b2) is the oil painting result using 45° as its stroke direction guidance; (c1) is the visualization of a random vector field; (c2) is the oil painting result using this random vector field as its stroke direction guidance. Observing (a2)(b2)(c2), the effectiveness of ETF reflects in two aspects. On the one hand, ETF guides the strokes along the tangent of the edges, which helps to produce sharp edges and clear details. For example, the bird's beak in (a2) has much sharper edges than in (b2)(c2), and the bird's eye in (a2) has more accurate details than in (b2) and (c2). On the other hand, ETF can capture the texture orientation features, which helps to preserve the content's semantic information. For example, the strokes of the bird's feathers in (a2) follow the feathers' actual growth direction, while the strokes in (b2) and (c2) cannot reflect the feathers' orientation feature. Both of these illustrate that using ETF could make the oil painting results more aesthetic.

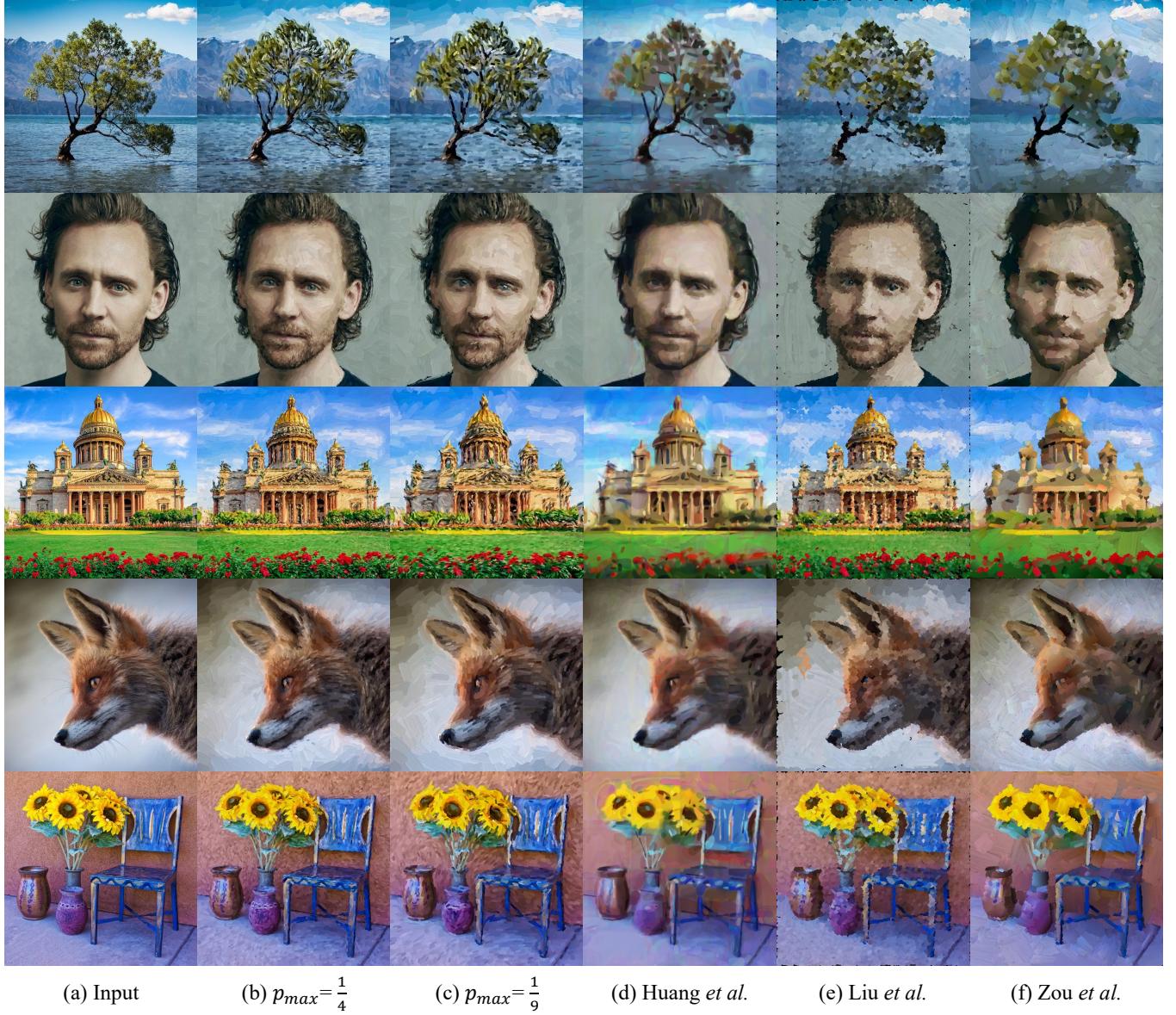


Figure 8: Oil painting comparison with three state-of-the-art methods: Huang *et al.* [23], Liu *et al.* [36], and Zou *et al.* [60]. Column (a) is the input. Column (b) and (c) are our results using $p_{max} = \{\frac{1}{4}, \frac{1}{9}\}$, respectively. Column (d) is the result of Huang *et al.* [23]; (e) is the result of Liu *et al.* [36]; (f) is the result of Zou *et al.* [60]. Please zoom in to compare details.

4.2 Fineness Control

We use the hyper-parameter maximum sampling probability p_{max} to control the fineness of our oil painting explicitly and linearly. As shown in Figure 7: (a1) is the input image; (a2) is the probability density map; (b1)(c1)(d1)(e1)(f1) are the oil painting results using $p_{max} = \{\frac{1}{4}, \frac{1}{9}, \frac{1}{16}, \frac{1}{25}, \frac{1}{36}\}$, respectively; (b2)(c2)(d2)(e2)(f2) are the corresponding anchors of (b1)(c1)(d1)(e1)(f1), which samples $K = \{5097, 2265, 1274, 815, 566\}$ anchors, respectively. Actually, the anchor number K is proportional to p_{max} due to the normalization operation in Section 3.1. We use value $\{\frac{1}{4}, \frac{1}{9}, \frac{1}{16}, \frac{1}{25}, \frac{1}{36}\}$ for p_{max}

because the maximum sampling density corresponds to one anchor in a square window with the side length of $\{2, 3, 4, 5, 6\}$, respectively. Therefore, the fineness of the oil painting changes linearly from (b1) to (f1), which can be seen from the glasses and the hair. More clearly, the fineness is proportional to $p_{max}^{-\frac{1}{2}}$. Using $p_{max} = \frac{1}{4}$ could produce fine-grained results while using $p_{max} = \frac{1}{36}$ leads to a very abstract result. For users who don't know the technical details, we just provide these five fineness levels for them to choose from, which is enough to meet the needs of application scenarios. Besides, it should be noted that our fineness control method is content-free,

which means the fineness only depends on the parameter p_{max} , regardless of the input image. This is because p_{max} determines the maximum density of strokes, while the image content only affects the total number of strokes.

5 COMPARISON EXPERIMENT

5.1 Qualitative Comparison

We compare our method with three state-of-the-art SBR oil painting techniques. As shown in Figure 8: column (a) is the input image; column (b)(c) are our results using $p_{max} = \{\frac{1}{4}, \frac{1}{9}\}$, respectively; (d)(e)(f) are the results of Huang *et al.* [23], Liu *et al.* [36], and Zou *et al.* [60], respectively. The results of the compared methods are generated by their official codes with default parameters.

Huang *et al.* [23] use the deep reinforcement learning algorithm DDPG (Deep Deterministic Policy Gradient) to train an oil painting agent. The main weakness of this method is that the results are too blurred and lack style since its oil painting agent is trained by minimizing the L_2 loss. Besides, this method uses Bézier curves as its stroke model and merges the strokes by transparency (alpha channel), both of which are inconsistent with actual oil painting, making its textures look unreal, and the overlapping areas appear ghosting-like artifacts. Compared with Huang *et al.* [23], our oil paintings are distinctive in style and have no artifacts. Liu *et al.* [36] devise a self-training network to produce oil paintings by predicting the parameters of a stroke set. The main weakness of this method is that the results look too rough, failing to form sharp edges. Most of the strokes are square and fragmentary, making the results like mosaics. In addition, this method somehow cannot fill up the entire canvas. There exist a lot of uncolored small black areas, especially at the borders. Compared with Liu *et al.* [36], our method achieves higher fidelity, and we can guarantee to fill up the canvas by the padding method introduced in Section 3.3. Zou *et al.* [60] design a rasterization network with a shading network to generate oil paintings. Similar to Huang *et al.*, the main weakness of Zou *et al.*'s method is that the results look too blurred, failing to preserve tiny details. Moreover, the stroke boundaries exist many obvious noise points, which heavily degrade the visual quality. Compared with Zou *et al.* [60], we can produce more detailed results, and our texture looks cleaner and clearer.

5.2 Quantitative Comparison

To further prove the application value of our method, we organized a Mean Opinion Score (MOS) test to measure the user preference for the above oil painting methods. Firstly, we collected a gallery dataset with fifty input images, which can be divided into five sets: scenery, portrait, animal, building, and still life. Every set contains ten images. For each image in this dataset, we use the compared methods [23][36][60] and ours with $p_{max} = \{\frac{1}{4}, \frac{1}{9}\}$ to produce five oil painting results.

Next, we invited a total of 115 volunteers from our university to participate in the MOS test. To avoid opinion bias among these participants, they were from eight different majors, each in roughly equal numbers. The volunteers were asked to score the oil paintings based on their visual perception in the following manner: every input image and its five oil paintings were packaged together as a group for display; the five oil paintings were anonymous in every

group, and their sequence was shuffled in different groups; participants had to observe each group of oil paintings for at least 10 seconds before giving their opinion scores; the available score values were integers ranging from 1 to 10. Besides, three groups of oil paintings would appear twice in the MOS test. If all these three groups of oil paintings' score rankings differ in the first and the second time, then we have reason to suspect that this participant is unreliable. Opinion scores collected from unreliable participants would be discarded. By this means, we identified 7 unreliable participants among these 115 volunteers.

After all the participants gave their opinion scores, for every reliable participant, his/her scores were normalized to $mean = 7.5$ and $variance = 1.5$. Finally, we calculated the average score for each method on each image set. The average opinion scores of different methods are shown in Table 1. Set 1-5 correspond to scenery, portrait, animal, building, and still life, respectively. The last column is the average score over the whole dataset. In each column, the best score is in bold, and the second is underlined. As a result, ours using $p_{max} = \frac{1}{4}$ achieves the highest opinion score in each set and the whole dataset; Huang *et al.* [23] achieves the second in Set-3; Liu *et al.* [36] achieves the second in Set-4; Zou *et al.* [60] achieves the second in Set-2; ours using $p_{max} = \frac{1}{9}$ achieves the second in Set-1, Set-5, and the whole dataset. In the last column, the mean scores of different methods have apparent gaps. Overall, users behave more preference to our oil paintings than the results of the compared techniques. The dataset and the corresponding oil painting results can be found in our [supplementary material](#).

Table 1: MOS test. Set 1-5 correspond to scenery, portrait, animal, building, and still life, respectively. In every column, the best score is in bold and the second is underlined.

Methods	Set-1	Set-2	Set-3	Set-4	Set-5	Mean
Huang <i>et al.</i> [23]	6.91	7.42	<u>7.60</u>	6.88	7.06	7.17
Liu <i>et al.</i> [36]	7.32	6.58	6.74	<u>7.47</u>	6.69	6.96
Zou <i>et al.</i> [60]	7.12	<u>8.07</u>	7.28	7.25	7.58	7.46
Ours, $p_{max} = \frac{1}{9}$	7.98	7.61	7.44	7.37	<u>7.84</u>	<u>7.65</u>
Ours, $p_{max} = \frac{1}{4}$	8.46	<u>8.27</u>	8.20	8.03	<u>8.34</u>	8.26

6 CONCLUSION AND FUTURE WORK

In this paper, we rethink the SBR oil painting problem from the perspective of adaptive stroke anchor sampling rather than the common pixel-wise approximation technique route. The anchor density is determined by the local texture complexity; the anchor number is determined by the global texture complexity; the anchor position is optimized by the Voronoi algorithm. By adjusting the hyper-parameter maximum sampling probability, we can control the oil painting's fineness from concrete to rough in a linear manner. Visual comparison with state-of-the-art SBR oil painting techniques demonstrates that our method achieves higher fidelity and better texture quality. The MOS test shows users behave more preference to our oil paintings than the results of other methods. In the future, we will try to extend our sampling-based method to more SBR styles, such as watercolor painting and pencil sketching. We may also explore the SBR style transfer problem.

REFERENCES

- [1] Franz Aurenhammer. 1991. Voronoi Diagrams—a Survey of a Fundamental Geometric Data Structure. *ACM Comput. Surv.* 23, 3 (Sept. 1991), 345–405. <https://doi.org/10.1145/116873.116880>
- [2] Christopher M Bishop et al. 1995. *Neural networks for pattern recognition*. Oxford university press.
- [3] Cassidy J Curtis, Sean E Anderson, Joshua E Seims, Kurt W Fleischer, and David H Salesin. 1997. Computer-generated watercolor. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, 421–430.
- [4] Oliver Deussen, Stefan Hiller, Cornelius van Overveld, and Thomas Strothotte. 2000. Floating Points: A Method for Computing Stipple Drawings. *Computer Graphics Forum* 19, 3 (2000). <https://doi.org/10.1111/1467-8659.00396>
- [5] Gianpiero Di Blasi and Giovanni Gallo. 2005. Artificial mosaics. *The Visual Computer* 21, 6 (2005), 373–383.
- [6] Gershon Elber and George Wolberg. 2003. Rendering traditional mosaics. *The Visual Computer* 19, 1 (2003), 67–78.
- [7] Geisa Martins Faustino and Luiz Henrique de Figueiredo. 2005. Simple adaptive mosaic effects. In *XVIII Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI'05)*. IEEE, 315–322.
- [8] Adam Finkelstein and Marisa Range. 1998. Image mosaics. In *Electronic Publishing, Artistic Imaging, and Digital Typography*, Roger D. Hersch, Jacques André, and Heather Brown (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 11–22.
- [9] Hongbo Fu, Shizhe Zhou, Ligang Liu, and Niloy J. Mitra. 2011. Animated Construction of Line Drawings. In *Proceedings of the 2011 SIGGRAPH Asia Conference* (Hong Kong, China) (SA '11). Association for Computing Machinery, New York, NY, USA, Article 133, 10 pages. <https://doi.org/10.1145/2024156.2024167>
- [10] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. 2016. Image Style Transfer Using Convolutional Neural Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- [11] Bruce Gooch, Greg Coombe, and Peter Shirley. 2002. Artistic Vision: Painterly Rendering Using Computer Vision Techniques. In *Proceedings of the 2nd International Symposium on Non-Photorealistic Animation and Rendering* (Annecy, France) (NPAR '02). Association for Computing Machinery, New York, NY, USA, 83–ff. <https://doi.org/10.1145/508530.508545>
- [12] David Ha and Douglas Eck. 2018. A Neural Representation of Sketch Drawings. In *International Conference on Learning Representations*.
- [13] Paul Haeblerli. 1990. Paint by Numbers: Abstract Image Representations. *SIGGRAPH Comput. Graph.* 24, 4 (Sept. 1990), 207–214. <https://doi.org/10.1145/97880.97902>
- [14] Alejo Hausner. 2001. Simulating Decorative Mosaics. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*. Association for Computing Machinery, New York, NY, USA, 573–580. <https://doi.org/10.1145/383259.383327>
- [15] James Hays and Irfan Essa. 2004. Image and Video Based Painterly Animation. In *Proceedings of the 32th International Symposium on Non-Photorealistic Animation and Rendering* (Annecy, France) (NPAR '04). Association for Computing Machinery, New York, NY, USA, 113–120. <https://doi.org/10.1145/987657.987676>
- [16] Christopher G Healey. 2001. Formalizing artistic techniques and scientific visualization for painted renditions of complex information spaces. In *IJCAI*, Vol. 1. 371–376.
- [17] Aaron Hertzmann. 1998. Painterly rendering with curved brush strokes of multiple sizes. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, 453–460.
- [18] Aaron Hertzmann. 2001. Paint by relaxation. In *Proceedings. Computer Graphics International 2001*. IEEE, 47–54.
- [19] Aaron Hertzmann. 2003. A Survey of Stroke-Based Rendering. *IEEE Computer Graphics and Applications* 23, 04 (2003), 70–81.
- [20] Aaron Hertzmann. 2009. Stroke-based rendering. *Recent Advances in Npr for Art & Visualization* 1 (2009).
- [21] Stefan Hiller, Heino Hellwig, and Oliver Deussen. 2003. Beyond stippling—Methods for distributing objects on the plane. In *Computer Graphics Forum*, Vol. 22. Wiley Online Library, 515–522. Issue 3.
- [22] Xun Huang and Serge Belongie. 2017. Arbitrary Style Transfer in Real-Time With Adaptive Instance Normalization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.
- [23] Zhewei Huang, Shuchang Zhou, and Wen Heng. 2019. Learning to Paint With Model-Based Deep Reinforcement Learning. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 8708–8717. <https://doi.org/10.1109/ICCV.2019.00880>
- [24] Bruno Jobard and Wilfrid Lefer. 1997. Creating evenly-spaced streamlines of arbitrary density. In *Visualization in Scientific Computing'97*. Springer, 43–55.
- [25] Henry Kang, Seungyong Lee, and Charles K. Chui. 2007. Coherent Line Drawing. In *Proceedings of the 5th International Symposium on Non-Photorealistic Animation and Rendering* (San Diego, California) (NPAR '07). Association for Computing Machinery, New York, NY, USA, 43–50. <https://doi.org/10.1145/1274871.1274878>
- [26] Henry Kang, Seungyong Lee, and Charles K. Chui. 2009. Flow-Based Image Abstraction. *IEEE Transactions on Visualization and Computer Graphics* 15, 1 (2009), 62–76. <https://doi.org/10.1109/TVCG.2008.81>
- [27] Hyung W Kang, Charles K Chui, and Uday K Chakraborty. 2006. A unified scheme for adaptive stroke-based rendering. *The Visual Computer* 22, 9 (2006), 814–824.
- [28] Dongyeon Kim, Minjung Son, Yunjin Lee, Henry Kang, and Seungyong Lee. 2008. Feature-Guided Image Stippling. In *Proceedings of the Nineteenth Eurographics Conference on Rendering* (Sarajevo, Bosnia and Herzegovina) (EGSR '08). Eurographics Association, Goslar, DEU, 1209–1216. <https://doi.org/10.1111/j.1467-8659.2008.01259.x>
- [29] Junhwan Kim and Fabio Pellacini. 2002. Jigsaw Image Mosaics. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques* (San Antonio, Texas) (SIGGRAPH '02). Association for Computing Machinery, New York, NY, USA, 657–664. <https://doi.org/10.1145/566570.566633>
- [30] Sung Y Kim, Ross Maciejewski, Tobias Isenberg, William M. Andrews, Wei Chen, Mario Costa Sousa, and David S. Ebert. 2009. Stippling by Example. In *Proceedings of the 7th International Symposium on Non-Photorealistic Animation and Rendering* (New Orleans, Louisiana) (NPAR '09). Association for Computing Machinery, New York, NY, USA, 41–50. <https://doi.org/10.1145/1572614.1572622>
- [31] Dmytro Kotovenko, Matthias Wright, Arthur Heimbrecht, and Bjorn Ommer. 2021. Rethinking Style Transfer: From Pixels to Parameterized Brushstrokes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 12196–12205.
- [32] David H. Laidlaw. 2001. Loose, artistic" textures" for visualization. *IEEE Computer Graphics and Applications* 21, 2 (2001), 6–9.
- [33] Liang Lin, Kun Zeng, Han Lv, Yizhou Wang, Yingqing Xu, and Song-Chun Zhu. 2010. Painterly Animation Using Video Semantics and Feature Correspondence. In *Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering* (Annecy, France) (NPAR '10). Association for Computing Machinery, New York, NY, USA, 73–80. <https://doi.org/10.1145/1809939.1809948>
- [34] Thomas Littemeier, Jens Metzner, Lena Pollak, and Oliver Deussen. 2015. Hardware-Based Non-Photorealistic Rendering Using a Painting Robot. In *Computer graphics forum*, Vol. 34. Wiley Online Library, 311–323.
- [35] Peter Litwinowicz. 1997. Processing Images and Video for an Impressionist Effect. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '97)*. ACM Press/Addison-Wesley Publishing Co., USA, 407–414. <https://doi.org/10.1145/258734.258893>
- [36] Songhua Liu, Tianwei Lin, Dongliang He, Fu Li, Ruifeng Deng, Xin Li, Errui Ding, and Hao Wang. 2021. Paint transformer: Feed forward neural painting with stroke prediction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 6598–6607.
- [37] Zhanping Liu, Robert Moorhead, and Joe Groner. 2006. An Advanced Evenly-Spaced Streamline Placement Algorithm. *IEEE Transactions on Visualization and Computer Graphics* 12, 5 (2006), 965–972. <https://doi.org/10.1109/TVCG.2006.116>
- [38] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137.
- [39] Aidong Lu, C. Morris, D.S. Ebert, P. Rheingans, and C. Hansen. 2002. Non-photorealistic volume rendering using stippling techniques. In *IEEE Visualization, 2002. VIS 2002*. 211–218. <https://doi.org/10.1109/VISUAL.2002.1183777>
- [40] Lei Ma, Yanyun Chen, Yinling Qian, and Hanqiu Sun. 2018. Incremental Voronoi sets for instant stippling. *The Visual Computer* 34, 6 (2018), 863–873.
- [41] David John Cameron Mackay. 1998. Introduction to monte carlo methods. In *Learning in graphical models*. Springer, 175–204.
- [42] Oliver Mattausch, Thomas Theußl, Helwig Hauser, and Eduard Gröller. 2003. Strategies for Interactive Exploration of 3D Flow Using Evenly-Spaced Illuminated Streamlines. In *Proceedings of the 19th Spring Conference on Computer Graphics* (Budmerice, Slovakia) (SCCG '03). Association for Computing Machinery, New York, NY, USA, 213–222. <https://doi.org/10.1145/984952.984987>
- [43] O.M. Pastor, B. Freudenberg, and T. Strothotte. 2003. Real-time animated stippling. *IEEE Computer Graphics and Applications* 23, 4 (2003), 62–68. <https://doi.org/10.1109/MCG.2003.1210866>
- [44] Mike Salisbury, Corin Anderson, Dani Lischinski, and David H Salesin. 1996. Scale-dependent reproduction of pen-and-ink illustrations. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. 461–468.
- [45] Michael P. Salisbury, Sean E. Anderson, Ronen Barzel, and David H. Salesin. 1994. Interactive Pen-and-Ink Illustration. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '94)*. Association for Computing Machinery, New York, NY, USA, 101–108. <https://doi.org/10.1145/192161.192185>
- [46] Michael P. Salisbury, Michael T. Wong, John F. Hughes, and David H. Salesin. 1997. Orientable Textures for Image-Based Pen-and-Ink Illustration. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '97)*. ACM Press/Addison-Wesley Publishing Co., USA, 401–406. <https://doi.org/10.1145/258734.258890>
- [47] Adrian Secord. 2002. Weighted-Voronoi Stippling. In *Proceedings of the 2nd International Symposium on Non-Photorealistic Animation and Rendering* (Annecy, France) (NPAR '02). Association for Computing Machinery, New York, NY, USA, 37–43. <https://doi.org/10.1145/508530.508537>
- [48] Robert Silvers. 1996. *Photomosaics: putting pictures in their place*. Ph.D. Dissertation. Massachusetts Institute of Technology.

- [49] Zhengyan Tong, Xuanhong Chen, Bingbing Ni, and Xiaohang Wang. 2021. Sketch Generation with Drawing Process Guided by Vector Flow and Grayscale. *Proceedings of the AAAI Conference on Artificial Intelligence* 35, 1 (May 2021), 609–616. <https://ojs.aaai.org/index.php/AAAI/article/view/16140>
- [50] Greg Turk and David Banks. 1996. Image-guided streamline placement. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. 453–460.
- [51] T. Van Laerhoven, J. Liesenborgs, and F. Van Reeth. 2004. Real-time watercolor painting on a distributed paper model. In *Proceedings Computer Graphics International, 2004*. 640–643. <https://doi.org/10.1109/CGI.2004.1309281>
- [52] David Vanderhaeghe, Pascal Barla, Joëlle Thollot, and François X Sillion. 2007. Dynamic point distribution for stroke-based rendering. In *Eurographics Symposium on Rendering*. Eurographics Association, 139–146.
- [53] Georges Winkenbach and David H. Salesin. 1994. Computer-Generated Pen-and-Ink Illustration. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '94)*. Association for Computing Machinery, New York, NY, USA, 91–100. <https://doi.org/10.1145/192161.192184>
- [54] Georges Winkenbach and David H Salesin. 1996. Rendering parametric surfaces in pen and ink. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. 469–476.
- [55] Ning Xie, Hirotaka Hachiya, and Masashi Sugiyama. 2013. Artist agent: A reinforcement learning approach to automatic stroke generation in oriental ink painting. *IEICE TRANSACTIONS on Information and Systems* 96, 5 (2013), 1134–1144.
- [56] Ning Xie, Tingting Zhao, Feng Tian, Xiao Hua Zhang, and Masashi Sugiyama. 2015. Stroke-based stylization learning and rendering with inverse reinforcement learning. In *Twenty-fourth international joint conference on artificial intelligence*.
- [57] Kun Zeng, Mingtian Zhao, Caiming Xiong, and Song Chun Zhu. 2009. From image parsing to painterly rendering. *ACM Trans. Graph.* 29, 1 (2009), 2–1.
- [58] Ningyuan Zheng, Yifan Jiang, and Dingjiang Huang. 2019. StrokeNet: A Neural Painting Environment. In *International Conference on Learning Representations*.
- [59] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. 2017. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*. 2223–2232.
- [60] Zhengxia Zou, Tianyang Shi, Shuang Qiu, Yi Yuan, and Zhenwei Shi. 2021. Stylized Neural Painting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 15689–15698.

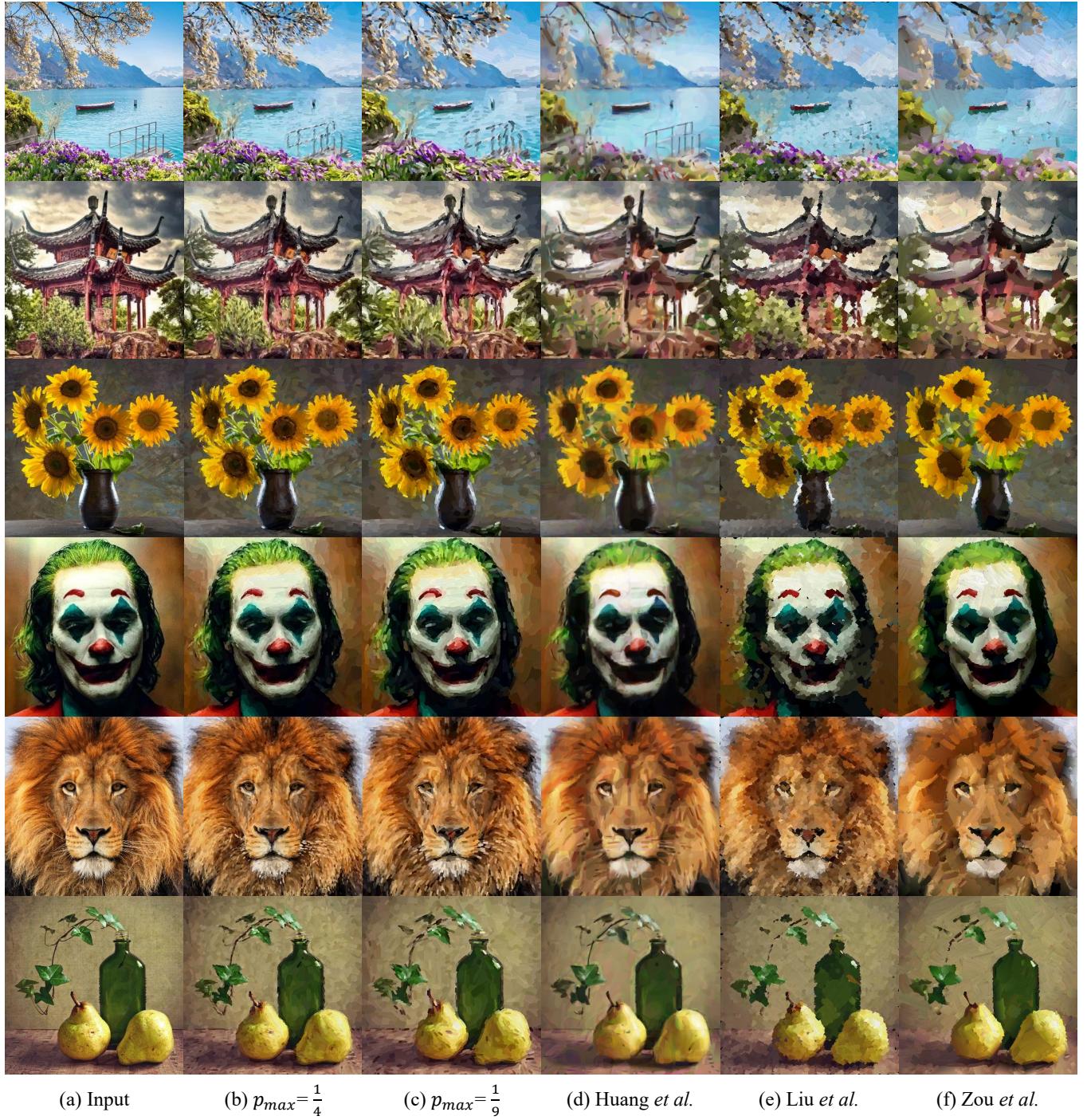


Figure 9: Oil painting comparison with three state-of-the-art methods: Huang *et al.* [23], Liu *et al.* [36], and Zou *et al.* [60]. Column (a) is the input. Column (b) and (c) are our results using $p_{max} = \{\frac{1}{4}, \frac{1}{9}\}$, respectively. Column (d) is the result of Huang *et al.* [23]; (e) is the result of Liu *et al.* [36]; (f) is the result of Zou *et al.* [60]. Please zoom in to compare details.

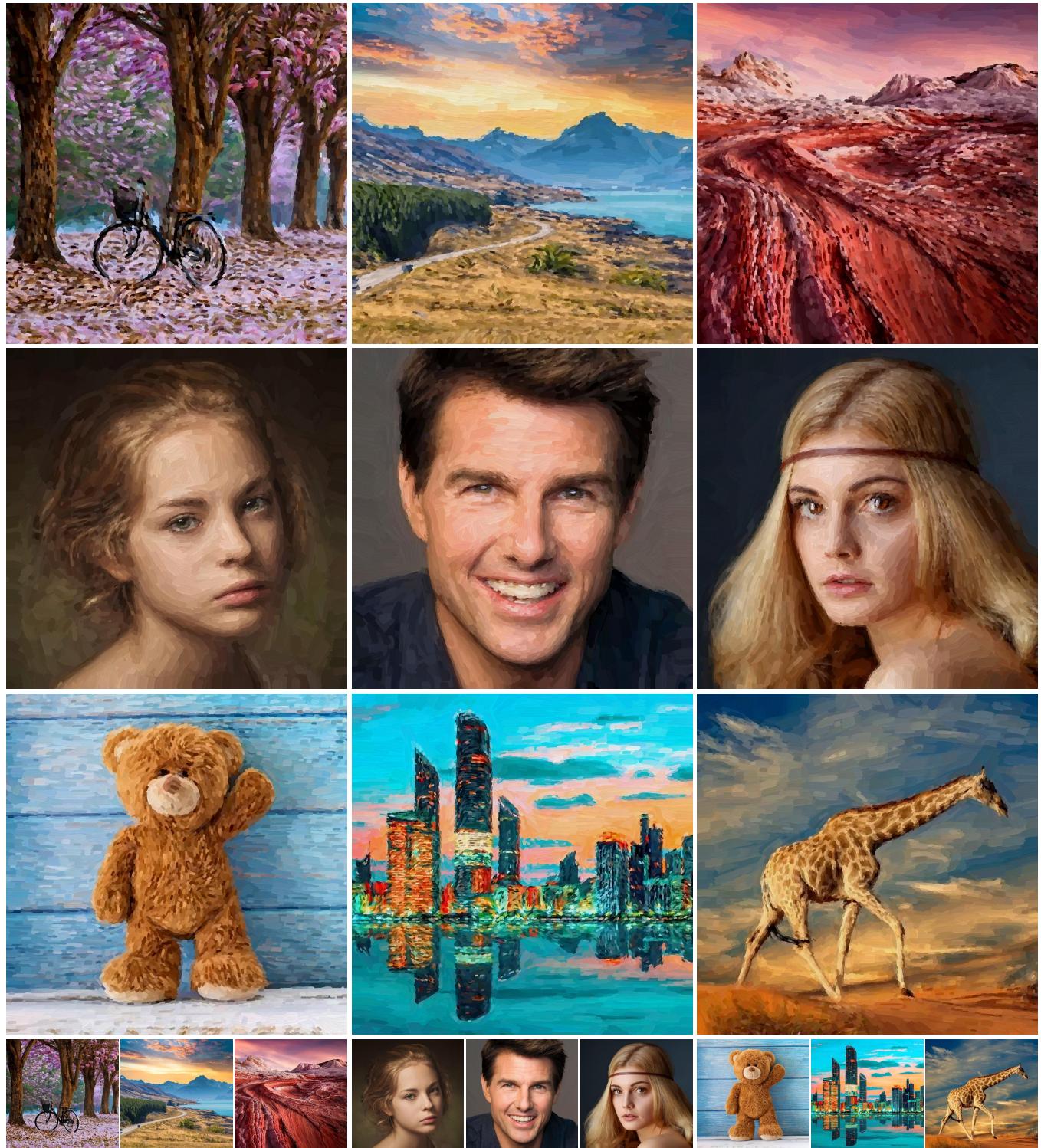


Figure 10: Nine oil paintings produced by our method using $p_{max} = \frac{1}{4}$. The input images are at the bottom.