Customizing Painterly Rendering Styles Using Stroke Process

Zijun Wei

https://github.com/zijunwei/GraphicsProject

Abstract. In this project we explore stroke based rendering(SBR), which is one of the fundamental research topics in Non-Photorealistic Rendering. We explore SBR by re-implementing the paper[1]. We created a user-friendly stroke based rendering software using OpenGL and Microsoft Foundation Class(MFC) Library that produces images with human-like painting styles.The major novelty of our software is that it enables user to intuitively control the output style. More than just following the pipeline proposed in [1], we also improved the stroke sampling and saliency computation methods used in[1] to produce more visual amusing visual effects. Our experiments show that our implementation is not only robust to input pictures of different input sizes but also capable of processing user-input efficiently and conveniently.

Keywords: Stroke Based Painterly Rendering, Non-Photorealistic Rendering, Painterly Rendering

1 Introduction

Simulating the common practice of human painters who create paintings with brush strokes is a hard problem because it is an subjective process that can be affected by many factors. Emotions, expressive styles and more are keys for the visualization of the painting work but they hard to model mathematically or to control automatically. Moreover, the barrier between artistic feeling and numerical parameters used in computer graphics make it difficult for users to create the desired styles.

In order to bridge the gap between expressive styles and computer graphics technology, we adopt eight intuitive parameters proposed by [1] that users can control to achieve the desired painting styles. Compared to parameters such as stroke distribution that is hard for users with no graphics background to understand, the proposed eight parameters are intuitive controls since they are widely used in perceptual dimensions. The eight parameters, which are proposed in [1], are listed below:

- Density: Stroke density is proportional to the number of strokes inside a unit image area.
- Non-Uniformity: The degree of unevenness of the spatial density of strokes. A high non-uniformity level means strokes are very dense in some places but very sparse elsewhere.

- Local Isotropy: The degree of similarity of stroke orientations in a neighborhood inside an image region. A high local isotropy level means neighboring strokes are usually near-parallel, exhibiting a smoothed style with low contrast in orientation.
- Coarseness: The average size of strokes. Generally, the larger the stroke sizes are, the coarser the rendered painting image is.
- Size Contrast: The local variance of size, represented by the size differences between each stroke and its neighboring strokes.
- Lightness Contrast: The differences in lightness of color between each stroke and its neighbors.
- Chroma Contrast: The differences in chroma of color between each stroke and its neighbors.
- Hue Contrast: The differences in hue of color between each stroke and its neighbors.

For our project, in addition to **a full implementation** of the method that is proposed in [1], we further improve the stroke process in the following aspects:

- We sample strokes non-uniformly based on Weibull distribution instead of 1D histogram matching on image grids. Weibull distribution elegantly incorporated the **density** and **non-uniformity** parameters in the non-uniform sampling process. It also avoids the information loss and the problem that samples might be overlapped on edges of grids in [1].
- Instead of simply computing saliency map of the input image by edge and ridge detection using steerable filters[2], we use a spectral residual approach[3] to detect image saliency that better approximate human vision mechanism.

This paper is organized as follows: section 2 explores the literature related to our stroke painterly rendering algorithm; section 3 goes through the basic components building up our algorithm; in section 4 we show experimentally that our proposed implementation produces high quality painterly rendered images; we conclude our work in section 5.

2 Related Work

Commonly researchers in this field model the stroke based rendering problem into two main steps: **brush modeling** and **stroke placement**. Research in both of these fields has achieved encouraging progress. We review these work in section 2.1 and section 2.2. Saliency map is closely related to stroke based rendering for it emphasizes on human's interest, which is an important subjective element to consider in painterly rendering, we review the work of saliency modeling in section 2.3

2.1 Brush modeling

For brush modeling, Strassmann [4] is among the earliest to study painterly graphical elements. After that, various improved methods [5–12] have been proposed. Currently it is considered as a solved problem. In this paper we simply

adopt the example-based method of [11], and use a dictionary containing around 6 textured brush strokes.

2.2 Stroke placement

For stroke placement, there are greedy and optimization-based methods [13]. In a greedy strategy, at each step, the algorithm determines the current stroke according to certain objectives and image/semantic features. The optimization-based methods compute the entire sequence of strokes together to achieve optimal global energies or desired statistics. Theoretically, optimization-based methods have the potential to outperform greedy ones, since they can explicitly model the interactions among strokes. These interactions, or high-order statistics among the strokes, essentially control the spatial contrasts mentioned above. Our method belongs to the optimization-based class, and it improves previous work in two main aspects. (1) It has a parameter design which explicitly emphasizes contrasts or high-order statistics, while parameters in most previous methods only correspond to either individual strokes or global features thus lack the power to reflect effects such as complementary colors in neighboring strokes. (2) Our method decomposes the energies/statistics into separately optimized terms corresponding to different perceptual dimensions. This not only simplifies computation, making it much faster than joint optimization [14] and MCMC sampling [15], but also enables flexible and friendly user customization.

2.3 Saliency Modeling

A good model of image saliency is a necessary for a successful painterly rendering because image saliency indicated where the strokes should be placed to emphasize human interests. Researchers have explored different methods to model image saliency for the past decades[16–19]. The saliency map provides a better description of key regions in images rather than simply using edge detectors [20] or human gaze [21] directly to model image saliency.

The literatures on saliency detection contain nearly 65 vision attention models in the last 25 years. We explain here why in our paper we use a spectral residual approach [3] to model image saliency. The reason is that first of all, compared to learning based method that takes large amount of training data and time, this is simpler, more efficient with comparable results. Second, experiment shows spectral residual approach strike a balance between high-level saliency features(i.e. human faces) and low level saliency features(i.e. color contrast).

3 Painting Algorithm

In this section we present our painting algorithm. Our painting algorithm accepts a single 2D image and eight intuitive parameters described in section 1, and produces a 2D painterly rendering of that input image. There are **five** stages to our algorithm, which in execution order are:

1. Image preprocessing. The segmentation map, saliency map and orientation field of the input image are computed.
2. Stroke sampling based on saliency map. We sample strokes based on the saliency map of the input image. Generally speaking, the more salient a pixel is, the more likely the stroke is placed on that pixel.
3. Stroke graph construction. The strokes are asymmetrically connected based on their spatial relationships, producing a stroke neighborhood graph. Then it is updated iteratively using stochastic reaction diffusion [22] to smooth the neighboring properties.
4. Stroke color and size properties are first of all initialized based on stroke positions and input image. Then they are updated using stochastic reaction diffusion in a similar way to the previous step by considering neighboring properties given stroke graph.
5. Brush models are picked form brush library and initialized accordingly based on stroke properties and finally placed on white canvas to produce a painterly rendering.

The whole algorithm is illustrated in Fig.1. Each step is detailed in the following chapter.

3.1 Image Preprocessing

In this step an input image is processed to generate the saliency map, segmentation map and orientation field.

Orientation Field For orientation field computation, we simply compute the gradient orientation of each pixel.

Segmentation Map For the segmentation map, we use the Simple Linear Iterative Clustering (SLIC) propose in [23]. It is considered as the state of art unsupervised algorithm to segment superpixels. SLIC performs a local clustering of pixels in 5D space defined by the L, a, b values of the CIELAB colorspace and x, y coordinates of the pixels. SLIC generates superpixels based on similarity of the 5D representation of each pixel. The distance is defined as follows:

$$d_{lab} = \sqrt{(l_k - l_i)^2 + (a_k - a_i)^2 + (b_i - b_k)^2} \tag{1}$$

$$d_{xy} = \sqrt{(x_k - x_i)^2 + (y_k - y_i)^2} \tag{2}$$

$$D_{ik} = d_{lab} + d_{xy} * (m/S) \tag{3}$$

Where D_{ik} is the sum of the lab distance and the xy plane distane normalized by the grid intervals S.

The cluster process starts by find the pixel which is a local minimal on gradient map to avoid edges points. The mean of all pixels in the cluster is used as new cluster center. At the end of iteration, connectivity is enforced by relabeling disjoint segments with the labels of the largest neighboring cluster.

In our implementation we fix the ratio $\frac{m}{S}$ to 0.5 so that the interval is not affecting final clustering results.

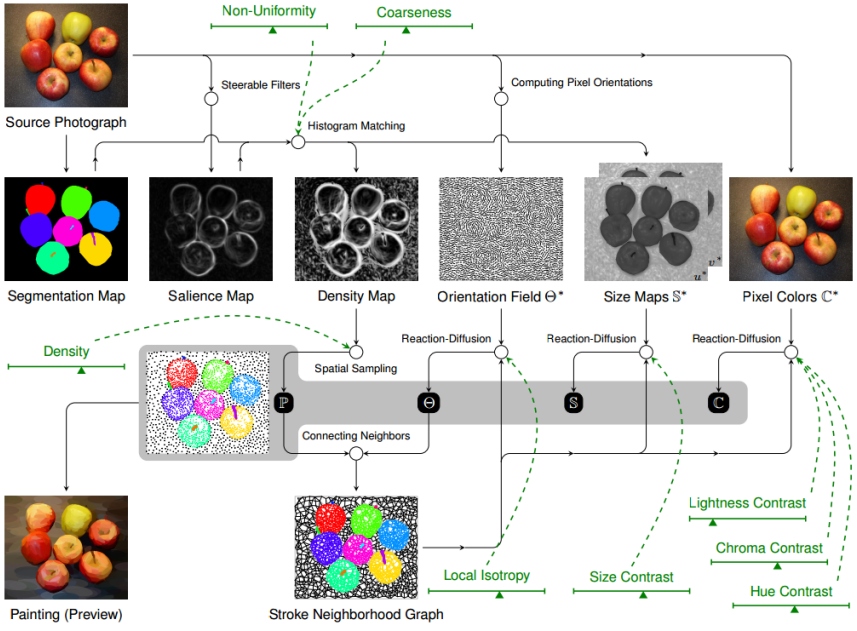

Fig. 1: The pipeline of the stroke processes. Green color and dashed arrows highlight the eight perceptual dimensions that users specify for each image region to the system (slidebars indicate their settings for the regions of apples). P, θ, S and C (black nodes in front of gray background) are the positions, orientations, sizes and colors of strokes to compute, respectively, with which we can render the final painting image or its fast preview. Gray segments in the stroke neighborhood graph (at the bottom) are connections between nodes in different image regions.
.

Saliency Map Instead of using only edge detector as indicators of salient regions of input image, we use the spectral residual algorithm proposed by [3]. According to Hou, information of the original image can be interpreted as the sum of the innovation and prior knowledge. The innovation stands for the interesting part with possible targets, whereas the prior parts stands for redundant or irrelevant information related to the background, where we will place fewer strokes.

Given an input image $I(x)$, the log spectrum $L(f)$ can be defined as

$$L(f) = log(F(I(x))) \tag{4}$$

where F represents the Fourier Transformation. Therefore, spectral residual $R(f)$ can be defined as:

$$F(f) = L(f) - A(f) \tag{5}$$

where $A(f) = h(f) * L(f)$ and $h(f)$ is a local averaging filter. Using inverse Fourier Transformation we can construct the saliency map in spatial domain.

3.2 Stroke Sampling

In this paper, instead of using the three-step stroke sampling method propose in [1]. We sample directly on pixels to avoid the error brought by grid approximation and possible overlapped sampling on grid edges. We first of all compute the probability for each pixel given by Weibull distribution:

$$F(x; k, \lambda) = 1 - e^{(x/\lambda)^k} \tag{6}$$

where x is the raw saliency value, and $k > 0$ and $\lambda > 0$ are the shape and scale parameters of the Weibull function. It can be seen form Equation 6 that λ naturally models the density and k models local iostropy(as opposed to 1D histogram mapping and region based random sampling).

We then sample the pixels with a probability given by the cumulative distribution function of Weibull distribution as in equation 6.

Note that we also did a non-maximum suppression to remove overlapped strokes. The threshold of overlap area is determined by the minimum size of strokes. With this step we can freely sample redundantly at first to ensure that the strokes will cover the whole canvas.

3.3 Stroke Graph Construction

Stroke Graph Initialization We initialize the stroke graph following [1]. Namely, we construct a Markov stroke neighborhood graph, whose nodes are the strokes at sampled positions, and edges connecting each node with up to four neighbors. we compute the neighborhood structure according to the distances between strokes and their orientations:

1. Initializing each strokes orientation θ to its reference value $\theta*$ in a reference orientation field prepared in advance. In our case the orientation field is initialized by the image gradient.
2. Constructing local two-dimensional Cartesian coordinates on each stroke. The origin is anchored at each stroke center, and the orthogonal straight lines $xy = 0$ are aligned with gradient direction and normal to gradient direction.
3. Connecting the four edges from the stroke to its nearest neighbor in each of the four quadrants. Nearest neighbors too far away (over a predefined distance threshold) are ignored, and strokes near region boundaries or image edges may not have neighbors in every quadrant (i.e., some neighbors may belong to other superpixels thus excluded from the neighborhood), so we allow less than four neighbors in such cases.

As soon as the stroke orientations are finally computed in the next step, the structure of the stroke neighborhood graph should be updated with refreshed neighborhood connections before we compute the other attributes.

An illustration of this algorithm is shown in Fig. 2

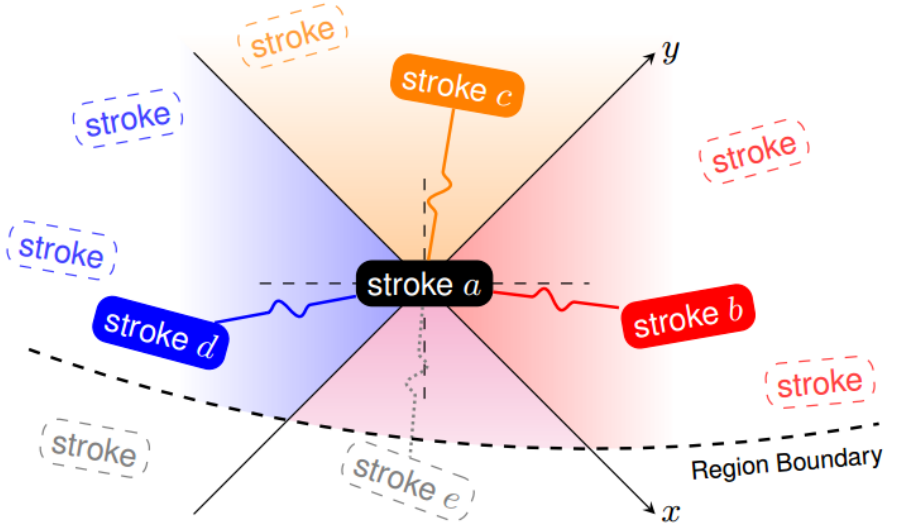

Fig. 2: Edge connections in the stroke neighborhood graph. A strokes neighborhood includes its nearest neighbor in each of the four quadrants, if there exists one within the predefined distance threshold inside the image region. In this figure, the neighborhood of stroke a is a set $N(a) = \{b, c, d\}$, and stroke e is excluded because it is not inside the same image region as a.

Stroke Graph Update We update the stroke graph by updating the orientation of each stroke and re-connect them under updated-orientations. As mentioned above, we update stroke graph before other properties because it also changes the graph topology.

We use stochastic reaction diffusion equations to update current stroke orientation based on neighboring properties, in which the diffusion smooths the attributes among neighboring strokes to reduce the contrasts (or enhances the contrasts if we use negative diffusion rates, as explained below), and the reaction preserves information from the source image. We detail below how we apply stochastic reaction-diffusion equations to update stroke orientations. For other properties, they use essentially the same form.

The stochastic reaction-diffusion is applied to orientation as follows:

$$\frac{d\theta}{dt} = R(\theta) + \lambda_\theta D(\theta) + \epsilon_\theta \tag{7}$$

The equation above is used to propagate information across the stroke neighborhood graph to compute the orientations iteratively, in which ϵ_θ is a small stochastic noise added to each iteration to simulate natural randomness. Since

θ is periodic over intervals of 2π, we adopt the orientation diffusion [24] term

$$D(\theta) = \sum_n w_n * sin(\theta_n - \theta) \tag{8}$$

where θ_n are orientations of neighboring strokes of the one currently being updated, and wn are weights inversely proportional to the spatial distances of these strokes. The local reaction term

$$R(\theta) = sin(\theta^* - \theta) \tag{9}$$

applies the persistent external force from the reference orientation field $\theta*$. The diffusion rate λ_θ is set to the level of local isotropy specified by user input.

After a few iterations θ is close to convergence, and we update the edge connections of the stroke neighborhood graph using the computed stroke orientations. The reaction-diffusion and graph updating are both very fast since the number of strokes is usually much smaller than that of image pixels.

3.4 Initialization and Update of Other Properties

The initialization and update of color(lightness, chroma, hue) and stroke size are basically the same as the preocess initialization and updating orientation. The reaction-diffusion of stroke colors includes two parts, for the aperiodic lightness and chroma, and the periodic hue, respectively. The process can be represented as follows:

$$dl/dt = (l^* - l) + \lambda_l \sum_n w_n(l_n - l) + \epsilon_l \tag{10}$$

$$dk/dt = (k^* - k) + \lambda_k \sum_n w_n(k_n - k) + \epsilon_k \tag{11}$$

$$dh/dt = sin(h^* - h) + \lambda_h \sum_n w_n sin(h_n - h) + \epsilon_h \tag{12}$$

The stroke size has 2 parameters: width and length. We initialize width and length based on stroke's gradient $[dx, dy]$. If dx is smaller than dy, width will be initialized to be the base size defined by coarseness while the length is dy/dx times the length of width and vice versa. The reaction-diffusion of stroke size is, however, a little bit more complex. In addition to the similar stochastic reaction diffusion precess:

$$ds/dt = (s^* - s) + \lambda_s \sum_n w_n(s_n - s) + \epsilon_s \tag{13}$$

We further restrict the size to be in a predefined range so that extreme situations will not happen. The reference size maps are generated from the salience map, but here the salience is firstly reversed (since intuitively, large salience corresponds to smaller brush stroke), the intuitive parameter coarseness is used to model the average size of strokes.

3.5 Brush Model Selection

When all the update are finished, the final step is to place the brushes on the canvas. Different from [1] that uses a commercial brush library of 200 brushes with varying shapes and texture appearances, which are collected from professional artists. These brushes are aimed at reflecting the material properties and feelings in several perceptual dimensions or attributes, for example, dry versus wet, hard versus soft, and long versus short, as well as four shape and appearance categories (point, curve, block, and texture). We do **NOT** have the access to the large brush library since they were commercialized. We created our own small library with 6 brushes.

In order to minimize the effect of inadequate number of brushes. We randomly select brushes from the library and resize them to the stroke size based on: (i) size contrast,(ii) the graident magnitude. The 6 brushes are visualized in Fig. 3

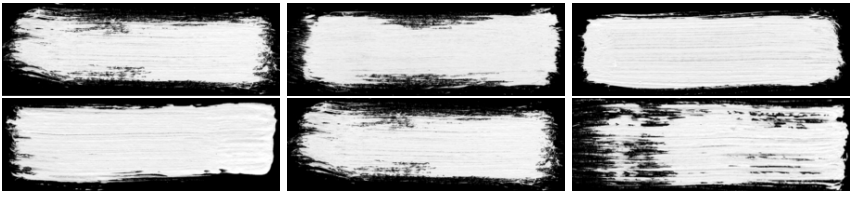

Fig. 3: Six brush models used in our algorithm

4 Experiments

4.1 Programming Language & Setups

This whole project is implemented in C++. The operating system is Windows 10. The development IDE is Visual Studio with MFC support. Only basic components of OpenCV such as image container and color conversion were used in our implementation. The image display pipeline was implemented using OpenGL for the sake of being familiar with graphics programming. **All the following steps, if not mentiond explicitly, were implemented by the author from scratch**.

All the visualizations are implemented using OpenGL. Visualization interfaces from OpenCV or Visual Studio were avoided. The OpenGL version of image visualization is implemented as follows:

Note that the parallel programming for placing strokes on image was not implemented in OpenGL because of time budget. It would be much faster if parallel stroke placement is used.

4.2 Experiment Parameters

We use input image with size around 400 by 400. In theory our algorithm can take any image size as input, the only difference is that it takes longer time to process the whole image.

Given an input image, several parameters needed to be pre-defined. The first one is the number of superpixels on the input image. We set the number of regions to be 20 so that dominant objects will be always be segmented out while avoiding the discontinuity of stroke graphs due to over-segmentation.

The second parameter the iterations for each reaction diffusion. We used the same parameters as used in [1]. Normally 100 iterations is a good number for all the properties.

The third parameter is the stroke size ratio. In order to avoid the misleading visual effect that the algorithm seems just simply blurs the image, we select the initial width height ratio to be 1:5.

There are also other parameters such as the scale of random noise added during stochastic reaction diffusion process, the distance threshold to decide if a stroke is a neighbor of current stroke and the rules to overlap brushes. They are relatively not detrimental to the final visual results. We simply use reasonable parameters in the context of input image size.

4.3 GUI

The GUI is shown in Fig. 4. The move of "density" and "non-uniformity" sidebar will lead directly to a visualization.

4.4 Qualitative Results

The intermediate process, namely, the stroke positions are visualized in Fig.**??** in order to show different density and non-uniformity parameters and their effects.

Some of the results are shown in Fig.6. As one may notice, the dog case is actually difficult because it requires many different level of details to view the eyes and so on. More examples can be seen in Fig .**??**

4.5 Timing

Given an input image of size around 400 by 400, on a Intel i5 dual core machine with 8G memory it is **roughly real time** to sample around 4K strokes using Weibull distribution without parallel programming. The total time from stroke initialization to all iterations done for all properties is roughly 5 seconds. There is high reason to do believe with proper parallel programming, the software will work in real time.

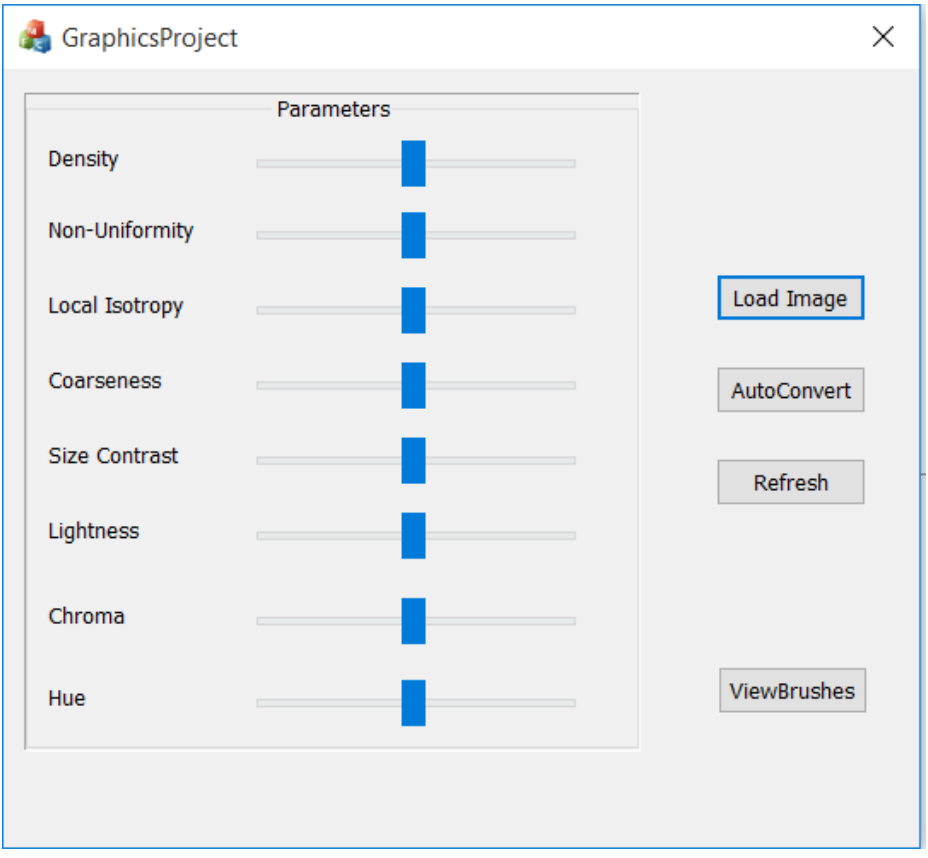

Fig. 4: GUI interface

5 Conclusion

We have implemented [1] with modification on saliency computation and control density sampling. Our implementation work well on given parameter settings in real time. There are some minor issues we want to address here and we plan to solve them in our future work. The first issue is that due to the large number of parameters used in this method, tuning work becomes difficult, especially the linear change of parameters may not be obvious in terms of results. One possible future work is to find the initial parameters automatically. Another future work is to implement the whole pipeline in parallel to make it real time.

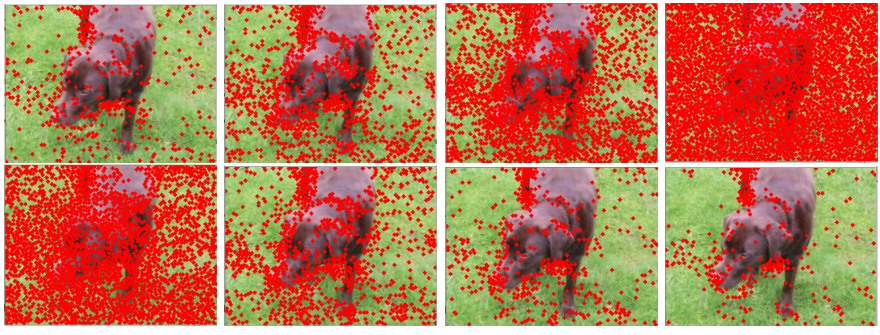

Fig. 5: Top row from left to right sampling from low density to high density. Bottom row from left to right: sampling from low non-unifomity to high non-uniformity

Fig. 6: Different parameter settings and different results. Top row: Difference in local isotropy; second row: difference in coarseness; third row: difference in size contrast; fourth row: difference in lightness. Best viewed in color.

Fig. 7: More results.

References

1. Zhao, M., Zhu, S.C.: Customizing painterly rendering styles using stroke processes. In: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering, ACM (2011) 137–146
2. Freeman, W.T., Adelson, E.H.: The design and use of steerable filters. IEEE Transactions on Pattern Analysis & Machine Intelligence (9) (1991) 891–906
3. Hou, X., Zhang, L.: Saliency detection: A spectral residual approach. In: Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on, IEEE (2007) 1–8
4. Strassmann, S.: Hairy brushes. In: ACM SIGGRAPH Computer Graphics. Volume 20., ACM (1986) 225–232
5. Cockshott, T., Patterson, J., England, D.: Modelling the texture of paint. In: Computer Graphics Forum. Volume 11., Wiley Online Library (1992) 217–226
6. Meier, B.J.: Painterly rendering for animation. In: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, ACM (1996) 477–484
7. Litwinowicz, P.: Processing images and video for an impressionist effect. In: Proceedings of the 24th annual conference on Computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co. (1997) 407–414
8. Hertzmann, A.: Painterly rendering with curved brush strokes of multiple sizes. In: Proceedings of the 25th annual conference on Computer graphics and interactive techniques, ACM (1998) 453–460
9. Hertzmann, A.: Fast paint texture. In: Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering, ACM (2002) 91–ff
10. Baxter III, W.V.: Physically-based modeling techniques for interactive digital painting. PhD thesis, Citeseer (2004)
11. Zeng, K., Zhao, M., Xiong, C., Zhu, S.C.: From image parsing to painterly rendering. ACM Trans. Graph **29**(1) (2009) 2
12. Chu, N., Baxter, W., Wei, L.Y., Govindaraju, N.: Detail-preserving paint modeling for 3d brushes. In: Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering, ACM (2010) 27–34
13. Hertzmann, A.: A survey of stroke-based rendering. IEEE Computer Graphics and Applications (4) (2003) 70–81
14. Hertzmann, A.: Paint by relaxation. In: Computer Graphics International 2001. Proceedings, IEEE (2001) 47–54
15. Hurtut, T., Landes, P.E., Thollot, J., Gousseau, Y., Drouillhet, R., Coeurjolly, J.F.: Appearance-guided synthesis of element arrangements by example. In: Proceedings of the 7th International Symposium on Non-photorealistic Animation and Rendering, ACM (2009) 51–60
16. Cheng, M., Mitra, N.J., Huang, X., Torr, P.H., Hu, S.: Global contrast based salient region detection. Pattern Analysis and Machine Intelligence, IEEE Transactions on **37**(3) (2015) 569–582
17. Li, H., Lu, H., Lin, Z., Shen, X., Price, B.: Lcnn: Low-level feature embedded cnn for salient object detection. arXiv preprint arXiv:1508.03928 (2015)
18. Liu, T., Yuan, Z., Sun, J., Wang, J., Zheng, N., Tang, X., Shum, H.Y.: Learning to detect a salient object. Pattern Analysis and Machine Intelligence, IEEE Transactions on **33**(2) (2011) 353–367
19. Judd, T., Ehinger, K., Durand, F., Torralba, A.: Learning to predict where humans look. In: Computer Vision, 2009 IEEE 12th international conference on, IEEE (2009) 2106–2113

20. Collomosse, J., Hall, P.: Painterly rendering using image salience. In: Eurographics UK Conference, 2002. Proceedings. The 20th, IEEE (2002) 122–128
21. Santella, A., DeCarlo, D.: Abstracted painterly renderings using eye-tracking data. In: Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering, ACM (2002) 75–ff
22. Zhu, S.C., Mumford, D.: Prior learning and gibbs reaction-diffusion. Pattern Analysis and Machine Intelligence, IEEE Transactions on **19**(11) (1997) 1236–1250
23. Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., Susstrunk, S.: Slic superpixels compared to state-of-the-art superpixel methods. Pattern Analysis and Machine Intelligence, IEEE Transactions on **34**(11) (2012) 2274–2282
24. Perona, P.: Orientation diffusions. Image Processing, IEEE Transactions on **7**(3) (1998) 457–467

Appendix : Major Implementation Blocks

1. A brush library that renders 6 brushes (myBrushes.cpp)
2. A MFC framework that enable GUI.(GraphicProjectDlg.cpp)
3. A stroke class that records current status of stroke(myStroke.cpp)
4. A status class recording current status of the storkelist(StrokeProcessState.cpp)
5. Weibull Sampling (NUS.cpp)
6. Image rendering 1: brush placement using OpenGL (painting.cpp)
7. Image rendering 2: Core OpenGL function for image loading and image showing. (glmy_showimage.cpp)
8. Parameter setting and converting to a reasonable range (parambox.cpp)
9. Saliency computation (computeSaliency.cpp)
10. Utility functions (utils.cpp)
11. Superpixel computation (Slicsuperpixel.cpp)
12. A bunch of visualization function for debug(vis.cpp)