Mr Painting Robot

Samuel Grant Dawson Williams, Richard Green

***Abstract*—This paper discusses the identification of outlines, and how this can be used for input into a creative painting algorithm. It builds on several existing algorithms including the bilateral filter and laplacian edge detection, and develops several new algorithms including intensity scanning and brush stroke formation. The primary benefit of this approach is that the output format is vector based (and thus suitable for robotic armatures), rather than pixel based (such as one would expect from the majority of painterly algorithms). Synthetic tests show that the entire process can reproduce sumi-e style images with over 90% accuracy.**

I. Introduction

Extracting important details from images is an interesting field of computer vision. The human vision system is very good at doing this, but computers don't have the body of knowledge required for complex classification or identification. Mr Painting Robot is an idea that a robot can be created to interpret and paint a scene creatively, including four primary elements: colours, outlines, shapes and textures.

A painting robot would ideally be a robotic arm with both a camera and a paint brush attached, and the ability to manipulate a wide variety of objects. The arm would use the camera to look around its environment and choose something to paint. The paints would be manually supplied to the robot in pots, and depending on the colours supplied, the painting algorithm would choose specific colours (potentially also depending on the mood of the robot). The robot could look at the object using its camera, and paint on the canvas using its brush.

This is quite a complex set of requirements, so this paper will primarily focus on the computer vision component of the problem rather than the mechanical and engineering issues.

II. Background

Painting is a fundamental form of expression which can consists of many discrete actions. Painting an image is a process (to some degree) of rationalising the human visual system into philosophical decisions and physical motions. In a sense, we interpret what we see, and then create something new using our experiences and aspirations.

The difficulty of establishing an empirical model for this process lies with the ambiguity of the task. Many different approaches have been studied[1], [2], [3], [4], [5], [6], [7], [8], [9]; there is no right or wrong way to paint a picture - it is a highly subjective act where the results may delight some and frustrate others. A simple thing such as the choice of colour, or the direction of a brush stroke could be the difference between perceived success or failure.

Fig. 1. The sample image.

We will be considering only a single colour - black ink and a single tool - a paint brush. This means that given any input image, the output must be a sequence of strokes, each stroke being made up of a sequence of connected points and associated thickness.

In order to interpret an image with these qualities, we need to consider the human mind and our ability to interpret the abstract.

A. Detecting Edges

There are many methods to extract shapes and lines from an image. In order to discuss several existing approaches, a sample image (see figure 1) will be used. There are several elements of this image which make it notable: significant textured and non-textured areas, same colour surfaces which change based on lighting conditions and surfaces with the same intensity value but differing colour.

The Canny algorithm[10] presents a computationally robust way of detecting edges in an image. This can be useful when we are interested in the shape of an object. When considering a painting algorithm however, too much information is lost. After processing an image using the Canny algorithm, we do not have any intensity or thickness information (see figure 2a).

The Hough transform[11] is a useful tool for identifying known shapes or lines in an image. It can be used to detect straight lines in an image, even when they lack continuity. However, it does not help us in the situation where we want to detect lines of unknown curvature, and so in this regard it is not suitable as an overall approach for extracting brush strokes.

The Laplacian operator[12] allows us to analyse an image for edges in a way that preserves thickness and intensity (see figure 2b). This gives a useful result for

(a) Canny edge detection (b) Laplacian edge detection (c) 3-Channel Laplacian edge detection

Fig. 2. Various edge detection algorithms.

Fig. 3. The bilateral algorithm applied to the original sample image.

Fig. 4. The grey scale conversion.

further analysis of such edges and their conversion to brush strokes.

To improve edge detection, we can firstly apply the bilateral filter[13] to the colour image (see figure 3). This filter reduces texture while maintaining the integrity of contrasting edges. This enhances the results produced by edge detection algorithms by reducing the amount of small details that show up.

In order to gain maximum information in our edge detection, we can consider each colour channel separately. The Laplacian operator is applied to separate channel, and then combined together to produce a colour image which outlines edges (see figure 2c). This gives us far more information about edge composition, such as edges that exist between colours of equal hue. A strong edge is white (detected in all three colour channels), and a weaker edge might only be detected in one colour channel.

B. Extracting Lines

When considering a potential brush stroke, we start from a single point and extend the stroke based on direction, thickness and curvature. We cannot rapidly adjust these properties, and so we need to apply some cost function to the search space of brush strokes. This is a slightly different goal from existing image vectorisation techniques[14], [15], [16], [17].

The Snakes algorithm[18] provides a potentially unique approach of fitting a line to a contour. One option would be to randomly generate brush strokes, and then use this algorithm to fit them to the processed image. This option would potentially enhance the artistic appeal of the final result.

The A*[19] algorithm can also be used to compute connectivity information. However, it requires a given start and end point for heuristic cost calculation, and can also have significant performance issues as it back-tracks through the search space.

A greedy depth-first search algorithm makes the best local decision at each step based on a cost function, and does not back-track. This ensures the performance of the algorithm in the worst case. It also means that the algorithm might not produce an optimal result.

C. Artistic Interpretation

In an abstract art form, pertinent shapes and objects are often emphasised, while background shapes and context may be ignored completely. The human visual system is supremely well adjusted to the environment that we exist in and thus classification is not an arduous task; however computer vision has little intrinsic ability to classify objects in a scene and to rank their overall importance. Semantic classification of objects in a scene can be used for brush stroke selection and other stylistic decisions[20], but this is not a primary focus of this research. Therefore, we will avoid any kind of local classification and simply focus

on producing brush strokes in a given image accurately over the entire image.

In art painted using only black, we need to make a particular distinction of contrasting artistic elements. We have limited ability to separate objects based on final colour or texture. In this regard, contour, thickness and direction can separate lines, but this is not always obvious if lines overlap. Computer vision systems are typically unable to isolate one object from another without advanced knowledge and classification algorithms. Therefore in this respect we will simply pay particular attention to the form of important shapes, while ignoring irrelevant texture and detail as much as possible.

In all art work, a sense of depth can be created by the artist by understanding the relative positions of objects in a scene. Lighting and shadows pay a large part in establishing the correct interpretation of an artwork, as well as geometric effects such as occlusion. Some computer vision systems allow for the analysis of depth, but robust systems for analysing lighting and shadows do not yet exist. In general, we will rely on our analysis of the source image to provide sufficient visual cues for depth related phenomenon, however further interpretation could assist with isolating important visual elements such as shadows and continuous segments which have been occluded.

III. Method

A brusk stroke is a sequence of points, where each point specifies an x, y coordinate and the width of the brush stroke at that point. With this information, we can create a basic painting robot. Additional information such as colour can be computed from the source image.

A. Intensity Scanning

The image is initially processed by firstly applying the bilateral filter, and then applying the 3-channel Laplacian operator to extract edge information.

An intensity scanning algorithm performs a sequence of evenly distributed vertical and horizontal scans on the image. The distance between scanlines is dictated by R (the inverse resolution). Each pixel is tested for intensity using a normalised Euclidian distance function:

$$Intensity(R, G, B) = \frac{\sqrt{R^2 + G^2 + B^2}}{\sqrt{3}}$$

We then extract a set of ranges of high intensity (see figure 5). A range consists of the start of a high intensity pixels, and the length that this high intensity sequence continues for.

For each extracted range, a circle is constructed with centre point x, y from the middle of the range, and diameter being the length of the range. This circle is then checked for coverage by calculating the number of high intensity pixels.

$$\frac{PixelsInCircle(image, circle, tolerance)}{AreaOf(circle)}$$

Fig. 5. An example of the intensity scanning process with R = 5.

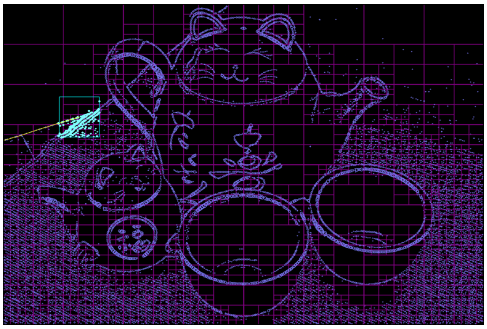


Fig. 6. An example of the quad-tree containing a set of intensity circles. An example of a line being formed can be seen.

If the coverage is not over a given minimum percentage, we reduce the size of the circle by a small amount and repeat this test. We continue until the coverage is satisfactory; we discard circles that become too small.

This set of circles is then representative of the final result.

B. Forming Lines

After processing the input as above, we have a set of circles covering areas of high intensity (see figure 6). We could simply paint each circle the appropriate colour, and we would be done. However, we wouldn't have achieved much more than a dot matrix printer. Joining the circles together to form lines allows us to start producing a meaningful result.

There are a number of criteria to consider depending on the kind of output format:

- We might need to reduce overlapping brush strokes. If the output is a wet ink - we can only cover the same spot a certain number of times.
- It might be difficult or unnatural to frequently move the brush at odd angles. Brush strokes potentially look better when they are straight.
- We might have a preference for longer strokes rather than short strokes. Many short brush strokes might create unwanted texture.

To build a line, we use a greedy search. We pick any circle at random, and then build a set of potential candidates for connection based on Euclidian distance. We

then feed these candidates through a cost function, and insert the results into a heap. After processing all possible candidates, we select the best choice from the top of the heap. We continue to build the line until there are no further candidates within a certain cost threshold.

After processing the line as far as we can in one direction, we reverse the line and continue processing. This has the effect that the line we build is as long as possible, in both directions from the starting point.

A quad-tree is used to improve the speed of space queries. It can also be used to perform pruning of the initial data-set, which can reduce the generation of lines that overlap and improve performance.

C. Cost Functions

The line generation algorithm is highly sensitive to the cost function used. There are several criteria calculating the cost between an existing line and a candidate point:

- We might want to try and keep the line as straight as possible.
- We might want to ensure that the line does not cross colour boundaries.
- We might want to ensure that the line does not cross low intensity edges.
- We might want to ensure that the line connects with the closest candidate.
- We might want to ensure that the thickness of the line does not change rapidly.

Based on these criteria, we can formulate many different cost functions. An example of a simple cost function is given (see figure 1).

Algorithm 1 $SimpleCost(image, line, point)$

```
cost ← 0
d ← DistanceBetween(EndOf(line), point)
if Size(Line) ≥ 2 then
    a ← AreaOfTriangle(LastSegment(line), point)
else
    a ← 1
end if
i ← IntensityBetween(EndOf(line), point, image)
return d × a + 1.0/i
```

D. Simulated Painting

After we have produced a set of lines, we can use this as the input to a robotic arm function which produces the appropriate motions on a canvas to produce an image. However, for lack of a suitable robot, we can simulate the results and use this for analytical testing.

Water colour[1] is difficult to simulate accurately. Thus, we will opt for the basic subtractive colour model with alpha accumulation and blending. This will be sufficient a creative reproduction.

Two brush stroke algorithms were implemented (see figure 7), one implemented by drawing a circle at each point and joining these together using polygons, and another using splines.

The spline brush stroke algorithm uses a stylised stroke where one end of the stroke is tapered to a point, and the other end ends with a rough finish. The purpose of this is to try and accurately simulate a simple brush.

There are several options for choosing colour (see figure 8). For a faithful Sumi-E rendition, we should use a strong black colour. Other options include processing the colour at each point along the brush stroke i.e. average, median, or point sampled colour.

As an experiment to increase the liveliness of the rendition, a semi-random colour boost model was implemented, where approximately every 3rd brush stroke would have the saturation and brightness of its colour boosted. This was done by converting the average colour into HSV, boosting the appropriate values, and then converting back into RGB for output.

E. Resolution Independence

Due to the nature of the output format, rasterisation can occur at any resolution. It is also possible to output the final result as a vector-based image.

IV. Results

Although the overall intrinsic result qualitatively appears good, the challenge is to quantify artistic perception.

A. Performance

The general performance of the algorithm is highly dependent on both the performance of space queries and the cost function (see table I). The initial implementation did not use a space partition and the performance was poor $O(N^2)$. The use of a quad-tree improves the general performance of the algorithm to $O(N \log N)$.

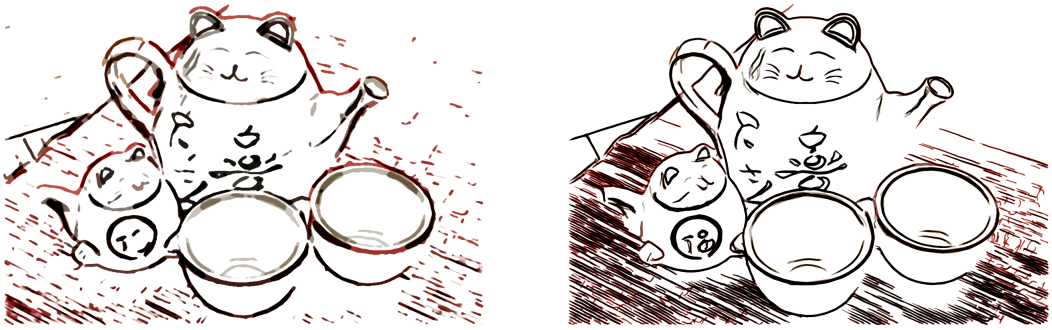
The resolution of the input image also affects the performance of the scanning algorithm, but not the stroke generation algorithm.

TABLE I
Execution Time

	Intensity Scan		Stroke Generation	
R	Circles Generated	Time	Strokes Generated	Time
1	40189	2.0s	2640	110s
2	20162	1.0s	1214	27s
4	10031	0.47s	570	8.6s
8	4995	0.27s	238	2.5s
16	2458	0.11s	67	0.67s

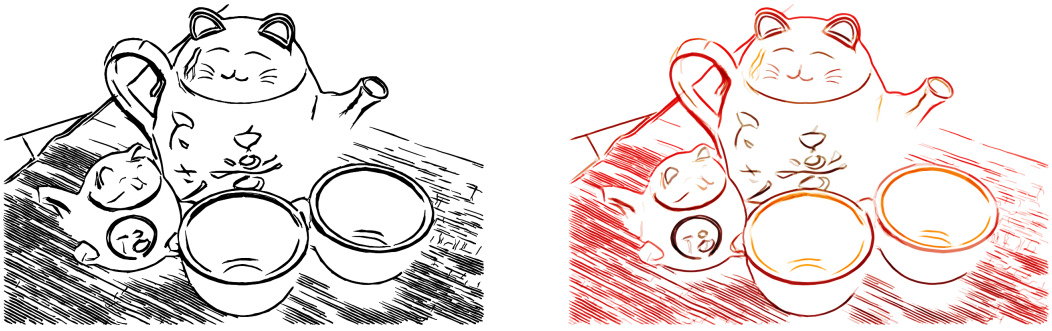
B. Accuracy

The general accuracy of the algorithm is fairly good (see table II). If we run the algorithm at $R = 2$ or $R = 4$ we get high accuracy as well as good performance. However, as we increase the accuracy, we also potentially get multiple strokes in the same location.



(a) Round Brush Strokes (b) Stylised Brush Strokes

Fig. 7. Various Brush Styles



(a) Sumi-e Colour Rendition (b) Colour Boost Rendition

Fig. 8. Various Colour Renditions

Accuracy can be measured by running the algorithm on a black and white input image, and stroking the output paths. We can then measure the ratio of correctly coloured pixels. In this case, we want to maximise both black and white accuracy.

TABLE II
Coverage Accuracy

	Spline Brush Accuracy		
R	White	Black	Difference
1	0.9858	0.9691	0.0167
2	0.9884	0.9648	0.0236
4	0.9904	0.9580	0.0324
8	0.9925	0.9347	0.0578
16	0.9940	0.8207	0.1733
	Circle Brush Accuracy		
R	White	Black	Difference
1	0.9871	0.9581	0.0290
2	0.9899	0.9473	0.0426
4	0.9913	0.9268	0.0645
8	0.9933	0.8893	0.1040
16	0.9946	0.7462	0.2484

V. Conclusion

The performance and accuracy of the algorithm are generally good. Sample images were processed in under a minute and produced artistic output. In the synthetic case, accuracies in excess of 90% were achieved. In general a set of brush strokes can be extracted from various kinds of input images, and this data is suitable for input to a robotic painting system.

All these qualities are improvements on previous work. Painterly algorithms produce image-space output[2], and vectorization algorithms don't provide output suitable for painting by a robot[16], [17].

A. Future Work

The output of this algorithm is suitable for input to a robotic function, but such a robot has yet to be constructed and programmed.

The algorithm could be enhanced to keep track of paint application in the case of overlapping lines. In some situations, it is desirable to reduce the number of overlapping brush strokes. This could be done by improving the quadtree pruning algorithm.

The output of the algorithm is highly dependent on the cost function. It is hard to conceive of an ideal cost function, and thus experimentation in this area could continue to yield improvements to accuracy and artistic value.

The algorithm currently uses input from an intensity image created using edge detection. However, this is not the only kind of input that could be used to create artistic

Fig. 9. A processed photograph of a Bonsai tree.

Fig. 10. A processed photograph of an indian Buddha.

imagery. Several approaches have been considered for future experimentation:

- Given n pots of different coloured paint, compute the Euclidian distance for each colour and use this as input to the painting algorithm.
- Use edge detection as currently implemented, but also fill in contiguous regions of specific colours for high intensity coverage (i.e. black).
- Use an image pyramid to detect large regions of consistent colour, and then use this information for 'washes' (large brush strokes which apply a small amount of colour).

References

[1] C. J. Curtis, S. E. Anderson, J. E. Seims, K. W. Fleischer, and D. H. Salesin, "Computer-generated watercolor," in *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, pp. 421–430. [Online]. Available: http://grail.cs.washington.edu/projects/watercolor/paper_small.pdf

[2] M. Shiraishi and Y. Yamaguchi, "An algorithm for automatic painterly rendering based on local source image approximation," in *NPAR '00: Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*. New York, NY, USA: ACM, 2000, pp. 53–58. [Online]. Available: http://w3.impa.br/~lvelho/ip/papers/npar2000.pdf

[3] A. Hertzmann, "Painterly rendering with curved brush strokes of multiple sizes," in *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 1998, pp. 453–460. [Online]. Available: http://mrl.nyu.edu/publications/painterly98/hertzmann-siggraph98.pdf

[4] G. Winkenbach and D. H. Salesin, "Rendering parametric surfaces in pen and ink," in *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 1996, pp. 469–476.

[5] ——, "Computer-generated pen-and-ink illustration," in *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 1994, pp. 91–100.

[6] M. P. Salisbury, M. T. Wong, J. F. Hughes, and D. H. Salesin, "Orientable textures for image-based pen-and-ink illustration," in *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, pp. 401–406.

[7] B. J. Meier, "Painterly rendering for animation," in *In SIGGRAPH 96 Conference Proceedings*, 1996, pp. 477–484.

[8] M. Obaid, R. Mukundan, and T. Bell, "Enhancement of moment based painterly rendering using connected components," in *CGIV '06: Proceedings of the International Conference on Computer Graphics, Imaging and Visualisation*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 378–383.

[9] S. C. Olsen, B. A. Maxwell, and B. Gooch, "Interactive vector fields for painterly rendering," in *GI '05: Proceedings of Graphics Interface 2005*. School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada: Canadian Human-Computer Communications Society, 2005, pp. 241–247.

[10] J. Canny, "A computational approach to edge detection," vol. 8, November 1986, pp. 679–698.

[11] R. O. Duda and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures," vol. 15, no. 1. New York, NY, USA: ACM, 1972, pp. 11–15. [Online]. Available: http://doi.acm.org/10.1145/361237.361242

[12] E. C. Hildreth, "Edge detection," September 1985. [Online]. Available: http://dspace.mit.edu/handle/1721.1/6429

[13] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *ICCV '98: Proceedings of the Sixth International Conference on Computer Vision*. Washington, DC, USA: IEEE Computer Society, 1998, p. 839. [Online]. Available: http://users.soe.ucsc.edu/~manduchi/Papers/ICCV98.pdf

[14] X. Hilaire and K. Tombre, "Robust and accurate vectorization of line drawings," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 6, pp. 890–904, 2006.

[15] P. Selinger, "Potrace," [Online; accessed June-2010]. [Online]. Available: http://potrace.sourceforge.net/

[16] T. Xia, B. Liao, and Y. Yu, "Patch-based image vectorization with automatic curvilinear feature alignment," *ACM Trans. Graph.*, vol. 28, no. 5, pp. 1–10, 2009.

[17] J. Sun, L. Liang, F. Wen, and H.-Y. Shum, "Image vectorization using optimized gradient meshes," *ACM Trans. Graph.*, vol. 26, no. 3, p. 11, 2007.

[18] A. W. Michael Kass and D. Terzopoulos, "Snakes: Active contour models," vol. 1, no. 4. Springer Netherlands, January 1988, pp. 321–331. [Online]. Available: http://webdocs.cs.ualberta.ca/~jag/papersVis2/levsetReadGr/snakesTerzopoulos.pdf

[19] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics SSC4 (2)*, pp. 100–107, 1968.

[20] K. Zeng, M. Zhao, C. Xiong, and S.-C. Zhu, "From image parsing to painterly rendering," *ACM Trans. Graph.*, vol. 29, no. 1, pp. 1–11, 2009.