Rendering Images in Painted Styles

Niall Williams*
Davidson College

Figure 1: *The Colosseum, Rome, Italy (left) and a painterly rendering of The Colosseum (right).*

Abstract

We present an implementation of a non-photorealistic rendering technique that renders images to look like painted images. The strokes of the painting are represented by cubic B-spline curves. The image is constructed in layers, starting with a rough sketch and then adding more detail with smaller brushes in subsequent layers. This allows the render to capture details in the image without the need for semantic analysis to identify important objects in the image.

Different styles of paintings can be produced by changing the program parameters. These parameters specify the number of layers and the minimum and maximum lengths of the strokes, which affects the level of detail in the rendered painting. We show that different input images should use different parameters to get the best results.

Keywords: image processing, painting, convolution, B-spline

1 Introduction

Rendering techniques often aim to create photorealistic images. Non-photorealistic rendering (NPR) focuses on rendering images with distinct artistic styles. The advantage of NPR is that it allows for more expressive, abstract renderings that photorealistic renderings cannot achieve. Different artistic styles can elicit different emotions from a user, meaning NPR can be used to more easily create the desired user experience for a graphical application. Depending on the application, NPR can also be used to avoid side-effects that may detract from the user experience. NPR can increase the performance of real-time applications by creating less complex scenes, and can help keep scenes out of the uncanny valley [MacDorman et al. 2009].

There are many different NPR techniques, including cel shading, Gooch shading, exploded view diagrams, scientific visualizations, and painterly rendering. This paper will focus on painterly renderings. *Painterly rendering* refers to converting a source input photo to a corresponding image that appears to be hand-painted [Geng 2011]. Painterly rendering algorithms allow us to easily create images in a variety of styles by simply adjusting algorithm parameters. This makes them useful for creating artwork at a very low cost, since, if the algorithm is good enough, a rendering will be just as good as a piece painted by an artist but will cost much less to create. Indeed, this ability to mass-produce painted images enables us to create films entirely in a painted style without needing to hand-paint each frame of the film [Hertzmann 2001]. It also gives users the ability to create paintings without needing to know how to paint–so long as the user has a source image and knows how the parameters influence the output, he or she can create paintings with the push of a button.

In this paper we present our implementation the painterly rendering algorithm created by Aaron Hertzmann [Hertzmann 1998]. We begin by providing an overview of different painterly rendering algorithms and assessing where these algorithms fall short. Next, we detail the entire painterly rendering pipeline. We then discuss the shortcomings of our implementation of Hertzmann's algorithm and provide potential ways to mitigate these shortcomings. We conclude with a brief overview of the project and a survey of future areas of related research. A gallery of sample renderings with different parameters is provided in the final section of the paper. Note that any assessments of an render's realism in this paper refer to how closely it mimics the appearance of a real painting of the same style, *not* how closely it resembles a photograph.

2 Background

The primary challenges when creating painterly renderings are creating strokes that have a natural texture and shape. Natural texture refers to how closely digital strokes mimic the physical appearance of real strokes painted in the same medium. Natural shape refers to how closely the digital strokes' paths mimic stroke paths of a real painting. Unnaturally shaped strokes will have very similar shapes and sizes. The algorithm developed by Hertzmann focuses primarily on stroke shape rather than texture [Hertzmann 1998].

*e-mail:niwilliams@davidson.edu

Algorithms have been successful at creating realistic paintings with different media including watercolor, pencils, and pen-and-ink [Curtis et al. 1997; Sousa and Buchanan 1999; Durand et al. 2001; Salisbury et al. 1997]. Additionally, aglorithms have been developed to create renders in particular styles including Chinese landscape paintings, mosaics, and stipples [Way and Shih 2001; Hausner 2001; Deussen et al. 2000]. These aforementioned algorithms often use short and simple strokes to automatically create images or require that the user places strokes which are then rendered accordingly. Using short strokes or having the user draw the strokes does not solve the problem of creating an algorithm that can fully automatically render realistic paintings.

The difficulty with creating realistic painterly renders mainly stems from the stroke placement. Art is a subjective discipline, and artists do not follow definitive rules when placing brush strokes. It is true that there are artistic principles that guide the general structure of paintings, but these principles do not dictate exactly where on the canvas an artist *must* place a stroke. Therefore, the challenge becomes creating a well-defined algorithm that places strokes according to strict rules (code) but does not appear too regular and unnatural. It is also important to note that this algorithm cannot be completely random, since art itself is *not* random. Artists' strokes are placed with reason rather than at random.

When considering algorithms that automatically create renders based on an input image, many of them suffer from unnatural stroke shapes. Some of these algorithms paint by placing jittered grids of short strokes across the image [Litwinowicz 1997; Treavett and Chen 1997]. Others produce images using relatively simple stroke structure, which contributes to the mechanic appearance of the renders [Shiraishi and Yamaguchi 2000]. Another reason why painting algorithms may produce unnatural images is because they paint the entire image in one pass, which eliminates the possibility for fine-tuning the render.

Hertzmann produced more painterly renders using two main concepts: (1) paintings are created in multiple layers, and (2) long, curved strokes can be represented with splines. Painting in layers mirrors the process a human artist will take when painting. The first layer consists of wide brush strokes that outline the general structure of objects in the scene. Later layers are created with fine brush strokes which add detail and capture the audience's attention, much like a real painting. This bottom-up layered approach also naturally divides the image into segments since it uses the reference image colors to know where to paint. Using splines to represent strokes allows the algorithm to avoid appearing mechanical. The curved strokes closely resemble real strokes and are easily calculated with a set of control points.

3 Painting Images

A high-level overview of the painting algorithm will now be discussed. The steps below are completed for every brush size, from largest to smallest.

1. Blur the original image with a Gaussian kernel with a standard deviation of $f_\sigma \times R$, where f_σ is a constant scaling factor and R is the current brush radius. The resulting blurred image is the reference image, I_r.
2. Identify regions of the reference image that should be painted on our canvas, I_c.
3. Create the strokes by following the normal of the reference image gradient. Do this for each region identified in step 2.
4. Paint all strokes for the current layer to the canvas, and repeat the process for the next smallest brush size.

3.1 Painting In Layers

One technique used by artists to mark objects as important to the scene is to paint these objects with more detail (other techniques utilize positioning, color, and shape. These are not discussed in this paper). See Figure 2. Painting with more detail requires finer brushes. To emulate this technique, Hertzmann's algorithm creates paintings layer-by-layer.

This iterative process naturally segments the image, since smaller brushes will more effectively highlight the differences in color at the edges of objects. It also naturally highlights the important objects in the painting. Large brushes will not be able to paint the detailed parts of an image, but they provide a rough outline of the primary shapes in the image. Details in the image appear when finer brushes are used, since they are able to capture smaller color differences in the reference image. See Figure 3.

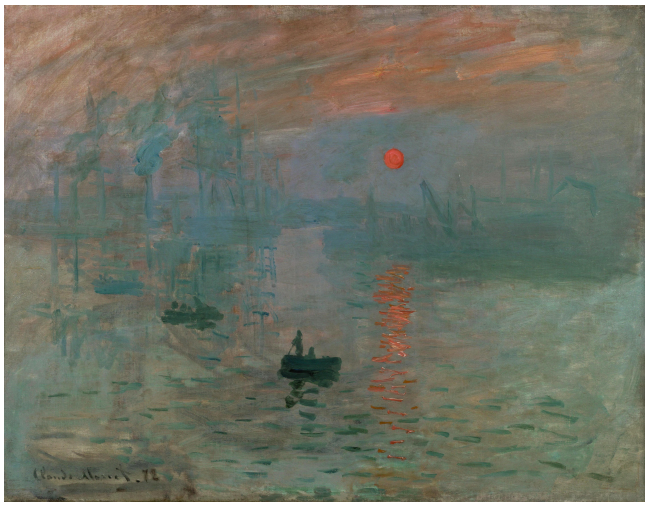


Figure 2: *Impression, Sunrise by Claude Monet. Monet draws attention to the sun's reflection and the boats by using fine brush strokes to paint these objects.*

Regions of high detail are automatically captured in the render because each layer has different intensities of blurring applied. Each brush can only capture the details that are at least as large as the brush size [Geng 2011]. In early layers, the reference image is highly blurred, which removes details from the image. Later levels have less blurring, so details are preserved in the reference image.

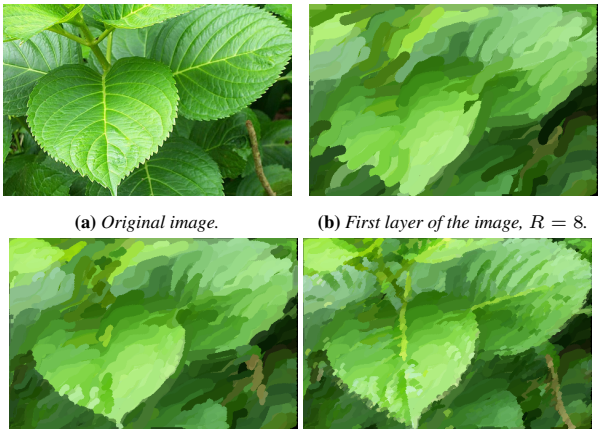
The blurring used in this algorithm is a Gaussian blur. We compute the blurred image by performing a convolution with a Gaussian kernel where $\sigma = f_\sigma \times R$.

3.2 Identifying Regions of Interest

Once the reference image is created, we need to locate the regions of I_r that need to be painted onto the canvas. To do this, a pointwise difference map $I_c - I_r$ is calculated. The difference between two images is the Euclidean distance of the pixel values,

$$|(r_1, g_1, b_1) - (r_2, g_2, b_2)| = ((r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2)^{1/2}$$

Using the difference map, the error in a neighborhood around each point is calculated. The error represents how different I_r is from I_c at pixel (x_i, y_i). Areas where the error is larger than some user-defined threshold are regions that should be painted. The size of the neighborhood around a pixel is given by a square window of



(a) *Original image.* **(b)** *First layer of the image,* $R = 8$. **(c)** *Second layer of the image,* $R = 4$. **(d)** *Third (final) layer of the image,* $R = 2$.

Figure 3: *Different layers of a painterly render. Fine lines like the midrib do not exist in the first and second layers, but do exist in the final layer.*

size $f_g R$ centered around the pixel of interest. f_g is some constant grid size factor specified by the user. The error of a neighborhood is calculated by,

$$\text{Area error} = \sum_{i,j \in N} \frac{D_{i,j}}{\text{grid size}^2}$$

where N is the pixel neighborhood and D is the difference map.

It is important to understand what I_r and I_c represent at this point. Suppose we are preparing to paint with the fourth brush in our list of brush sizes, so $i = 4$. In this scenario, the canvas has only been painted with brushes of radius $R_1, ..., R_{i-1}$. This means the amount of detail painted on the canvas is the maximum amount of detail that can be captured with a brush of radius R_{i-1}. We blur the original image by a factor of R_i, which is a smaller factor than all previous blur operations for this render. This means details previously lost in I_r are now present. Therefore, when we create a difference map between I_r and I_c, areas of high difference indicate areas where new details have appeared. This property of the algorithm is the reason we do not need to perform any semantic anaylsis to identify the important regions of the image. It relies on the idea that areas of interest in an image will be highly detailed.

Once areas of high difference are located, the strokes must be calculated and painted. Each stroke originates from the pixel of highest error in a high-error neighborhood. The color of the stroke is the color of the reference image at the first control point.

3.3 Creating Curved Strokes

Curved brush strokes are key to creating convincing painting renders. Long, curved strokes can convey shapes and expressions not possible with short strokes. According to Hertzmann, at the time of the original paper's publication, all other automatic painting algorithms use small strokes that are identical aside from color and orientation [Hertzmann 1998]. To render long, curved strokes, we model strokes as cubic B-splines and calculate the curve based on the spline's control points. The direction of a curve is based on the gradient of the luminance of I_r.

The first step to creating a stroke is to convert I_r to its luminance channel. The luminance of a pixel is calculated by [Hughes et al. 2014]:

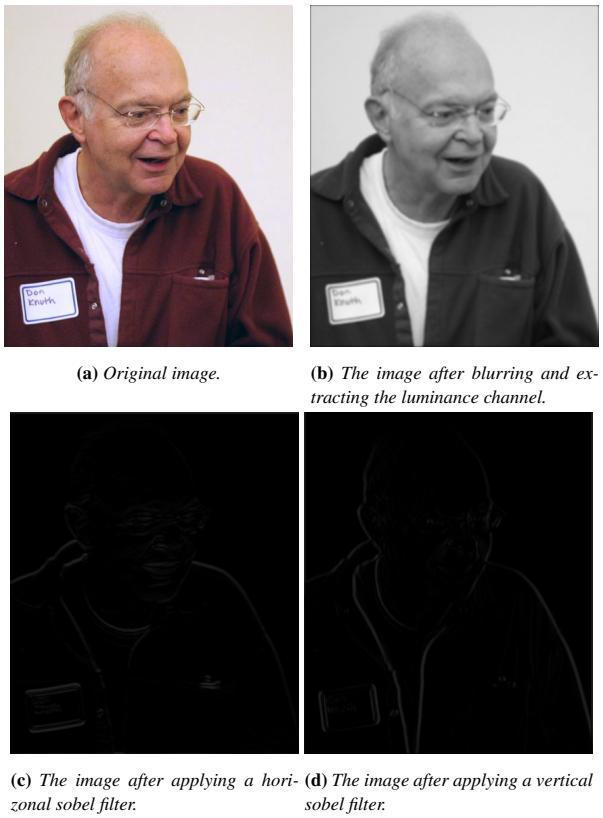
$$L(r, g, b) = 0.30 \times r + 0.59 \times g + 0.11 \times b$$

An example of an blurred image converted to its luminance channel can be seen in Figure 4b.

Next, we calculate the gradient of the image. The gradient of the image is the directional change in intensity in the image. For a grayscale image, the closer a pixel's color is to white or black, the larger the gradient. High gradient values denote edges in the image.

To calculate the image gradient, we convolve the grayscale image with a sobel filter kernel. The sobel filter requires that we calculate the gradient in the x- and y-directions. These can be seen in Figures 4c and 4d. When the sobel filter outputs a high value, it means an edge is at that pixel value.

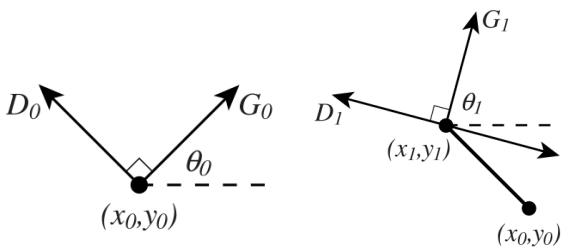
It is important to note that the sobel filter detects lines which may not necessarily be edges. For the purpose of image segmentation this is not ideal since the filter will highlight lines that do not separate two segments of an image. However, for painterly rendering, this is not a problem. We need to paint strokes not only along edges, but also within objects if we are to capture the appropriate details and textures of the objects.



(a) *Original image.* **(b)** *The image after blurring and extracting the luminance channel.* **(c)** *The image after applying a horizonal sobel filter.* **(d)** *The image after applying a vertical sobel filter.*

Figure 4: *Different stages of the image processing pipeline.*

We trace the stroke on a path perpendicular to the gradient, which allows us to paint a stroke along the edges of the image. The stroke's first control point is located at the pixel of highest error in a high-error neighborhood, which is described in Section 3.2. Each subsequent control point is placed in the direction perpendicular to the gradient at the last control point at a distance of R_i. There are two directions perpendicular to the gradient, so we choose the direction that minimizes the stroke curvature (the angle between D_i and $D_{i-1} \leq \frac{\pi}{2}$). See Figure 5.



(a) *Spline after placing one control point.*

(b) *Spline after placing two control points.*

Figure 5: ***(a)*** *The stroke originates at point (x_0, y_0) and the gradient is the vector G_0. The direction the next control point is placed is along the vector D_0.* ***(b)*** *After placing the next control point along D_0, the gradient G_1 at the new control point (x_1, y_1) is calculated. At this point, there are two directions perpendicular to G_1 so we choose D_1 since it reduces the spline curvature. This process is repeated for each control point. Image adapted from [Hertzmann 1998].*

We add control points to the spline until we have reached the maximum stroke length (defined by the user) or the difference between the stroke color and the color of the pixel at the last control point becomes too large. The user also defines a minimum stroke length which prevents strokes from becoming too short and regular.

3.4 Painting Strokes

Strokes for a layer are only painted once every stroke in that layer has been computed. Before painting, the stroke order is shuffled. If the stroke order is not shuffled, strokes will consistently overlap such that strokes closer to the top left corner of the image are occluded by strokes closer to the bottom right corner. In addition to preventing consistent occlusion, shuffling the strokes contributes to the natural appearance of the render. A comparison between shuffled and unshuffled strokes can be seen in Figure 6.

(a) *Strokes within a layer are painted in a random order.*

(b) *Strokes within a layer are painted in the order they are computed.*

Figure 6: *A comparison between renders in which the strokes are painted in a random and sequential order. Random order adds to the naturalism of the render and avoids regular stroke overlap.*

Each stroke is painted by calculating the curve defined by the stroke's control points. This curve is a cubic B-spline. Any stroke consists of only one color which is the color in the reference image at the first control point. After all strokes for a layer are painted, the entire process described in this section repeats for the next smallest brush size.

4 Discussion

4.1 Evaluation of Rendering Success

It is difficult to measure the quality of our results. To our knowledge, there is no scale that scores how painterly a rendering appears. The best way we can evaluate our results is to compare them to Hertzmann's results and real paintings.

We believe our results are good. It is no question that our renders look like paintings. The layered structure of our renders looks like that of Hertzmann's renders. We were able to create different styles by changing the program parameters, which helps us emulate real paintings even more. A good example of different painting styles can be seen in Figure 9.

Another strong point of our results is the irregularity of the strokes. Our strokes do not appear unnatural like they do in some other painterly rendering algorithms. Our strokes follow the image gradient closely enough that they are oriented differently according to the gradient, rather than all being oriented in the same direction.

Despite our success, there are some problems with our algorithm. While our strokes are not completely regular, there appear to be some common features of most of the strokes. In Figure 3b, most of the strokes are at a diagonal. This happens in almost all of our images. The strokes do have curves, but in general the end points of our strokes are closer than the start points to the top-right corner. Another area where our results do not excel is the sobel filter. Our sobel filtered images have a substantial amount of noise in them. We do not know the cause for this. It is possible that the regularity of our strokes has something to do with the sobel filtered images, since we use the sobel filtered images to guide our strokes. Finally, there is a bug in our program that sometimes causes strokes to converge to the top-left corner of the image. This can be seen in Figure 11. This bug occurs when we try to draw a spline that has less than four control points, since the definition of a spline curve requires at least four control points. Based on initial tests carried out after the code due-date, only drawing strokes with at least four control points seems to fix this bug.

4.2 Learning Outcomes

Through this project we learned about techniques for image processing and non-photorealistic rendering. Specifically, we learned how to use convolutions and splines to process and render images.

We learned how to combine different processing techniques to create a rendering algorithm. This algorithm makes clever use of Gaussian blurs and Euclidean distance to mimic the workflow a real artist follows. Identifying regions of interest through blurring and subtraction is an unexpected way to identify regions of detail in an image. It requires no semantic analysis to understand which sections of an image require more attention.

We also learned about developing algorithms for problems that initially may not seem to be solvable by an algorithm. Art is often considered to be very different to computer science, but this project shows that it still shares some algorithmic characteristics with computer science. This project is an example that seemingly unstructured systems can have structure and rules about them that can be translated into an algorithm that a program can easily execute. Careful observation of how a system, or parts of a system, works can reveal characteristics of the system that can be "exploited" to develop a working algorithm.

5 Conclusion

In this paper we presented an implementation of the non-photorealistic renendering algorithm developed by Aaron Hertzmann [Hertzmann 1998]. When given an image, this algorithm outputs the same image but with the appearance of a painting. We presented a detailed overview of the algorithm. This included details about the image processing pipeline, and the theory behind why the algorithm works. We also explained technical details about how to implement the algorithm.

This project is at the intersection of art and computer science. It serves as an example of the beautiful elegance of both art and computer science. Painting is a complex and impressive endeavor, and this algorithm attempts to break down this complex process in to simple steps. In fact, it succeeds at emulating real painting. The rich expression and delicacy of paintings are recreated with this algorithm using a relatively simple set of rules and procedures.

One avenue of future work is to optimize the current implementation. Our convolution kernels are implemented with 2D kernels even though both the Guassian and sobel kernels are separable into two 1D kernels. This is because we did not have enough time to implement 1D kernels. The original implementation also makes use of a z-buffer to avoid painting strokes that are occluded, which is a feature not included in our implementation.

Future work can also extend the program's capabilities. Hertzmann's implementation supports stroke transparency, which allows his renders to imitate a wider range of styles. The original implementation also adds random jitter (noise) to the strokes to avoid a totally uniform stroke appearance. It also renders anti-aliased strokes to further improve the render's faithfulness to real paintings. Finally, real-time rendering can allow us to create entire videos rendered in particular painting styles.

References

CURTIS, C. J., ANDERSON, S. E., SEIMS, J. E., FLEISCHER, K. W., AND SALESIN, D. H. 1997. Computer-generated watercolor. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 421–430.

DEUSSEN, O., HILLER, S., VAN OVERVELD, C., AND STROTHOTTE, T. 2000. Floating points: A method for computing stipple drawings. In *Computer Graphics Forum*, vol. 19, Wiley Online Library, 41–50.

DURAND, F., OSTROMOUKHOV, V., MILLER, M., DURANLEAU, F., AND DORSEY, J. 2001. Decoupling strokes and high-level attributes for interactive traditional drawing. In *Rendering Techniques 2001*. Springer, 71–82.

GENG, W. 2011. *The Algorithms and Principles of Non-photorealistic Graphics: Artistic Rendering and Cartoon Animation*. Springer Science & Business Media.

HAUSNER, A. 2001. Simulating decorative mosaics. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM, 573–580.

HERTZMANN, A. 1998. Painterly rendering with curved brush strokes of multiple sizes. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM, 453–460.

HERTZMANN, A. P. 2001. *Algorithms for rendering in artistic styles*. PhD thesis, New York University, Graduate School of Arts and Science.

HUGHES, J. F., VAN DAM, A., FOLEY, J. D., MCGUIRE, M., FEINER, S. K., SKLAR, D. F., AND AKELEY, K. 2014. *Computer graphics: principles and practice*. Pearson Education.

LITWINOWICZ, P. 1997. 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., 407–414.

MACDORMAN, K. F., GREEN, R. D., HO, C.-C., AND KOCH, C. T. 2009. Too real for comfort? uncanny responses to computer generated faces. *Computers in human behavior 25*, 3, 695–710.

SALISBURY, M. P., WONG, M. T., HUGHES, J. F., AND SALESIN, D. H. 1997. Orientable textures for image-based pen-and-ink illustration. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 401–406.

SHIRAISHI, M., AND YAMAGUCHI, Y. 2000. An algorithm for automatic painterly rendering based on local source image approximation. In *Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, ACM, 53–58.

SOUSA, M. C., AND BUCHANAN, J. W. 1999. Observational model of blenders and erasers in computer-generated pencil rendering. In *Graphics Interface*, vol. 99, 157–166.

TREAVETT, S., AND CHEN, M. 1997. Statistical techniques for the automated synthesis of non-photorealistic images. In *Proc. 15th Eurographics UK Conference*, 201–210.

WAY, D.-L., AND SHIH, Z.-C. 2001. The synthesis of rock textures in chinese landscape painting. In *Computer Graphics Forum*, vol. 20, Wiley Online Library, 123–131.

Gallery

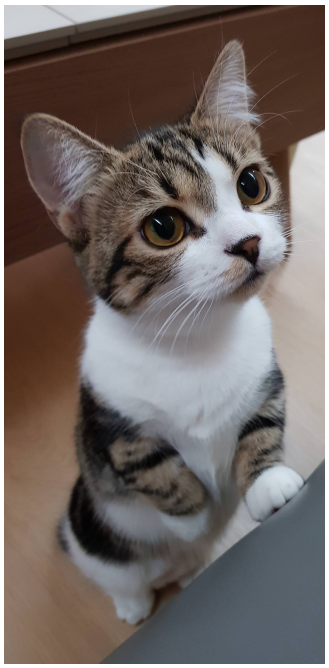
The rest of this paper features various images and their painterly renders. Different styles are created by using different stroke parameters.

(a) *Girl with a Pearl Earring by Johannes Vermeer, 1665.*

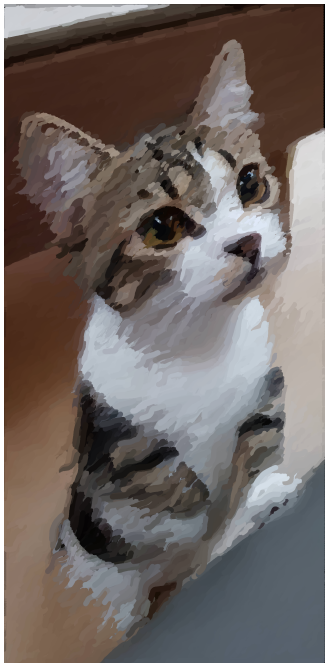
(b) *Scarlett Johansson as the Girl with a Pearl Earring.*

(c) *Painterly rendering of Scarlett Johansson.*

Figure 7: *A comparison of an original painting, a physical reenactment of the painting, and a painterly rendering of the reenactment.*

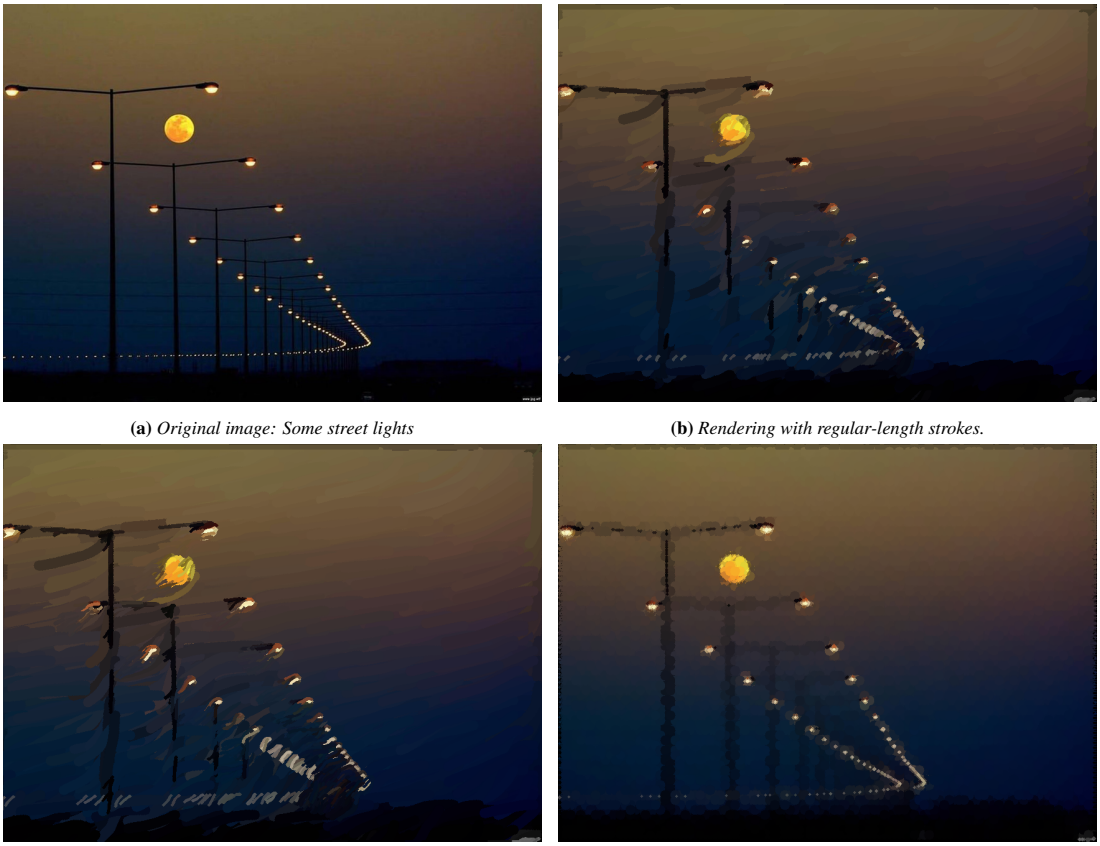


(a) *Original image: A cat standing up.*



(b) *The cat painted with brushes of sizes 12, 6, 3.*

Figure 8: *Example render.*



(a) *Original image: Some street lights*

(b) *Rendering with regular-length strokes.*

(c) *Rendering with long strokes.*

(d) *Pointillist style rendering.*

Figure 9: *A demonstration of different results generated by changing algorithm parameters. The halos of light around the bulbs are well-defined with a pointillist style, but are smeared and unclear when rendered with full strokes.*

(a) *Original image: A man in a skeleton costume.*

(b) *Rendering with medium- and large-length strokes.*

Figure 10: *Example render. Photo via Butch Loscin.*

(a) *Original image: Fathers and their children.* **(b)** *Rendering with medium- and large-length strokes.*

Figure 11: *Example render.*

(a) *Original image: Some mountains.* **(b)** *Rendering with medium- and large-length strokes.*

Figure 12: *Example render.*

(a) *Original image: A manatee.*

(b) *Rendering with medium- and large-length strokes and a high curvature filter, which produces more curved strokes.*

Figure 13: *Example render.*

(a) *Original, high-resolution image: Statues underwater.*

(b) *Rendering with 3 brushes of size 8, 4, 2. The input image is a very high resolution, so the fine brushes capture most of the detail and the final render looks very similar to the original image.*

Figure 14: *Example render.*

(c) *Rendering with 5 brushes of size 32, 16, 8, 4, 2. Larger brushes cause us to lose more details, but cause the render to more clearly look like a painterly render.*

Figure 14: *An example of the detail preserved and lost depending on the brush sizes used.*