Visual Comput (2007)
DOI 10.1007/s00371-007-0183-y

ORIGINAL ARTICLE

Chuan-Kai Yang
Hui-Lin Yang

Realization of Seurat's pointillism via non-photorealistic rendering

C.-K. Yang (✉) · H.-L. Yang
Department of Information Management
National Taiwan University of Science and Technology,
No. 43, Section 4, Keelung Road, Taipei, 10607, Taiwan
ckyang@cs.ntust.edu.tw,
D9509102@mail.ntust.edu.tw

Abstract Photorealistic rendering is one of the oldest and most important research areas in computer graphics. More recently, the concept of *non-photorealistic rendering* has been proposed as an alternative with important advantages for numerous application areas. The popularity of non-photorealism can be mainly attributed to its simplicity, which could potentially lead to a state of aesthetics and succinctness. Reality often presents too many details and too much complexity, thus offsetting the observation of the essence of objects, and objects' interaction with lights. Based on a similar belief, *impressionism* focuses primarily on conveying the interaction of light and shadows without emphasizing the fine details of a scene. In recent years, there has been a trend towards simulating impressionism with computers. Among the various styles of impressionism, we are particularly interested in simulating the style of *pointillism*, especially the style presented by *Georges-Pierre Seurat*, deemed the founder of pointillism. The reason his style attracts us is twofold. First, the painting process of pointillism is extremely laborious, so unloading the main proportion of the manual painting task is mostly desired. Second, though several existing general-purposed algorithms may be able to approximate pointillism with point-like strokes, some delicate features frequently observed in Seurat's paintings are still not satisfactorily reflected. To achieve simulating Seurat's painting style, we have made careful observations of all accessible Seurat's paintings and extracted from them some important features, such as the relatively few primitive colors, color juxtaposition, point sizes, and, in particular, the effects of complementary colors and halos. These features have been more successfully simulated and results are comparable with not only Seurat's existing paintings, but also with previous attempted simulations.

Keywords NPR · Pointillism · Halos · Complementary colors · Dithering

1 Introduction

Traditionally there is almost no doubt that computer graphics technologies aim at mimicking reality. Sophisticated mathematics and physics are involved during the pursuit of reproducing the reality through computers. Such an ambition has been nearly fulfilled in recent years due to the blossoming of techniques on *global illumination*, *texture mapping* and *image-based rendering*. Surprisingly, there was a different trend emerging in the last decade: how to perform rendering in *non-photorealistic* styles. The success of such a dramatically different philosophy is not accidental, but could be attributed to at least the following two reasons. First, just like what happened in art history,

both realism and abstractionism have their supporters, but if we regard the computer graphics technology only as one of the many means to create images, then photorealism needs not to be the only goal. Second, nowadays it is often deemed an era of information-explosion; therefore, modern data tend to become huge and more complex, or sometimes even beyond human comprehension, and this situation may get even worse with the consistent advances on the data-acquisition process. *Non-photorealistic rendering*, or *NPR* for short, may serve as an alternative means to convey information in a more simplified and succinct way, and examples of such applications can be found in [10, 13].

As one of the most famous non-photorealistic painting styles, *impressionism* concerns mainly the interaction between light and shadows, or what the so-called *impression* that a scene can bring to a person, without placing too much emphasis on irrelevant details. There has been a trend towards simulating impressionism with computers, either by converting existing images or videos into ones with impressionistic styles, such as [9, 12, 14, 19–21, 32] or by designing a graphical painting interface where images of impressionism styles could be easily composed, such as [5, 17, 18, 22]. Among the previously simulated impressionism styles, *pointillism* catches most of our attention, and among the painters in pointillism, Seurat attracts us for the following two reasons. First, it is well known that a pointillistic painting usually takes a very long time to finish, as it involves a very tedious process which is potentially suitable for computers. Second, though generic NPR techniques exist to approximate pointillism effects, such as [12, 14, 19, 20, 32], they are often too general to faithfully reflect all the features presented in Seurat's works. For example, in all Seurat's paintings, the number of colors in his color palette is very restricted, and some even said that essentially he used only *eleven colors* in his works. Furthermore, we observed that a feature called *halo*, that frequently appears in his paintings, has not been successfully simulated so far. Moreover, Seurat is also characterized by his use of *color juxtaposition* and *complementary colors*. To accurately emulate Seurat's painting style, we have made careful and detailed observations on numerous paintings of his in order to extract the common or important features that distinguish his work from others. With these features properly addressed, a given image, accompanied by a user's choice of several parameters, could be turned into a painting-like image with a presumably Seurat's style. To prove the success of our efforts, simulated images are presented at the end of the paper, together with their comparisons with the results produced by other existing techniques. One of Seurat's own paintings, after digitization, is also shown for being the ground truth for comparison.

The rest of the paper is organized as follows. Section 2 reviews some existing works related to this study. Section 3 briefly describes what we observed from Seurat's paintings, i.e., the distinct features. Section 4 discusses how we simulate Seurat's painting styles step by step. Section 5 presents our simulation results, which are compared against previous simulations and one of Seurat's paintings. Section 6 concludes this paper and hints for potential future directions.

2 Related work

The name of *non-photorealistic rendering* was first brought to the computer graphics community by Winkenback et al. in 1994 [30]. Since then, a huge number of papers have been generated. As our main goal is to convert existing images into pointillistic ones, we will only pay attention to those similar studies. Readers who are interested in learning more information regarding NPR could refer to [29].

Perhaps it would be better that we first distinguish between *stippling* [6] and pointillism. In stippling, the focus is mainly on the placement of points, where every point usually bears a similar outlook. To simulate an area with a darker shade, denser points will usually be disposed on that area; on the contrary, a brighter area normally corresponds to a lesser number of points. In pointillism, points could vary not only in sizes, shapes, but also in colors. We will explain how we achieve pointillism in great details in Sect. 4.

Probably being the first work to simulate pointillistic effects, Hertzmann proposed to mimic the hand-painted features by curved brush strokes, and in particular, he attempted some pointillistic effects by placing dense circles with perturbed hue and saturation values [14]. A layered structure is employed to process an image layer by layer, with progressively smaller brushes. While each layer is a blurred version of the original, smaller brushes are only applied to where the original image differs from the blurred one. Numerous results, including source images, are available, and will be compared with ours. Litwinowicz also proposed to simulate impressionism by varying the attributes of strokes [19]. In addition, he also gave discussions on more attributes of a point, such as clipping, anti-aliasing, texturing, orientation, and so on, so that users could have more control on the simulation results. Temporal coherence was also mentioned for dealing with videos. Similarly, Hays et al. also presented an elaborated scheme that could control various attributes of strokes and therefore lends itself to stylized images or videos [12]. Simulation results, including the originals, are also available for comparisons. Zhu et al. introduced a unified framework that could facilitate the control of stroke attributes and the inclusion of other features to be developed in the future [32]. Hertzmann proposed a rather different approach for simulating a stylized painting by performing *image analogies* [15]; however, we believe that his method cannot produce better results than that by a more domain-specific approach.

Most of the aforementioned approaches concentrate on simulating the strokes, while leaving color variation to a random perturbation. The work of Luong et al. [20] adopted a different approach, where they pick colors in an *isoluminant* manner, so that the results match well with the human perception system. Similarly, Jing et al. [16] divided an input image into tiles, and within which colors are also perturbed in an isoluminant way. The work by Hausner [11] is probably the most similar work to ours so far. However, the *half-toning* patterns that he generated by an *error diffusion* method seem too regular to create an artistic feeling. There is also not much variation in terms of point sizes and shapes, not to mention the lack of other important features, such as halos that frequently present in pointillistic paintings.

In general, nearly all of these approaches tried to address impressionism as a whole, without really digging into some specific features generally observed in most pointillistic paintings. This paper, though admittedly not with a high ambition, believes that each and every painting style posses its own uniqueness, which needs not to be "unifiable" with others. We, therefore, devote ourselves to none but one painting style: Seurat's pointillism.

3 Seurat's pointillism

Georges-Pierre Seurat (December 2, 1859–March 29, 1891) was deemed the founder of *neoimpressionism*, *pointillism*, or *divisionism*, and in fact each of these names has its own meaning. *Neoimpressionism* represents what he intended to do, *pointillism* describes which drawing primitive he chose, and *divisionism* explains how he achieved his goal. Neoimpressionism was proposed to challenge what *impressionism* had been doing, that is, paint by intuition and spontaneity. However, what Seurat believes was that a more rigorous or scientific approach should be adopted for conveying a desired scene. He later selected *point*, presumably the most basic painting primitive, as his weapon to fulfill what he wanted, and this also explains how pointillism got its name. Finally, through divisionism, or *point juxtaposition*, he managed to create the illusion of desired colors by a visual mixture of the colors from a more restricted set. In general, there are several noticeable features commonly observed in his paintings. As the study of these features has really motivated this work, we therefore first discuss each of these important features before plunging into the implementation details.

3.1 Eleven colors

Seurat was known for only using relatively few colors when composing his paintings. While some said that he used only eleven colors, so far to our knowledge, we cannot be sure of what these eleven colors are. However, the traditional rainbow colors are surely included as those colors were recognized long time ago. Some other intermediate colors directly mixed from these primitive colors should also be enclosed for the sake of completeness, with the exception of the *dirt* color, which is thought to be too "impure"; that is, the making of it involves the mixture of more than two colors.

3.2 Halo effect

Figure 1 shows a partial image of one of Seurat's most famous paintings, *Sunday afternoon on the island of La Grande Jatte*. From the partially enlarged portion, one could clearly see the *halo* effects that surround the lady's dark dress. Here we define the *halo* effect to be the contrast enhancement effect applied on and near the silhouette edges. This effect is mostly used to depict object boundaries.

Figure 2 is Seurat's another famous painting, *young woman powdering herself*. One can see that some crease edges, which are not silhouette edges, also get enhanced from time to time.

3.3 Complementary colors

Another important feature is to utilize complementary colors. The main purpose of such usage is to make the original colors more prominent after being contrasted with their complementary counterparts. Figure 1 gives a demonstration for this. Although it is well defined in modern color theory regarding the complementary color for a given color, we are not completely sure of what

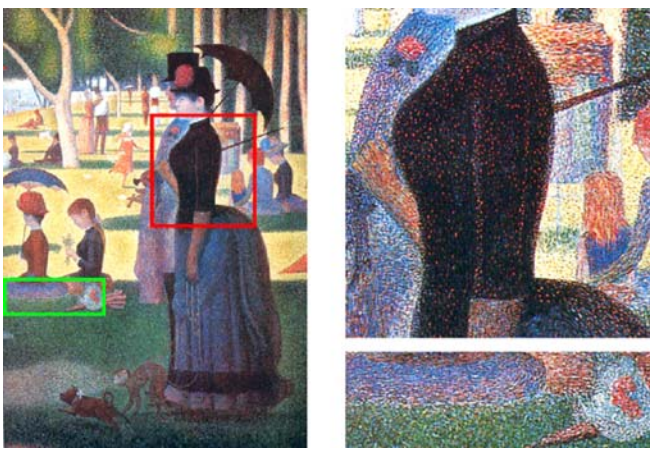

Fig. 1. *Left*: a partial image of Seurat's *Sunday afternoon on the island of La Grande Jatte* painting. *Upper-right*: an enlarged portion of the *left image* marked with *red*, where one can see how the halo (along the boundary of the lady's dress) and complementary colors (the *red dots* on the lady's dark dress) take place. *Lower-right*: another enlarged portion of the *left image* marked with *green*, where the *green color* is juxtaposed with yellow and blue, to make the *green color* stick out

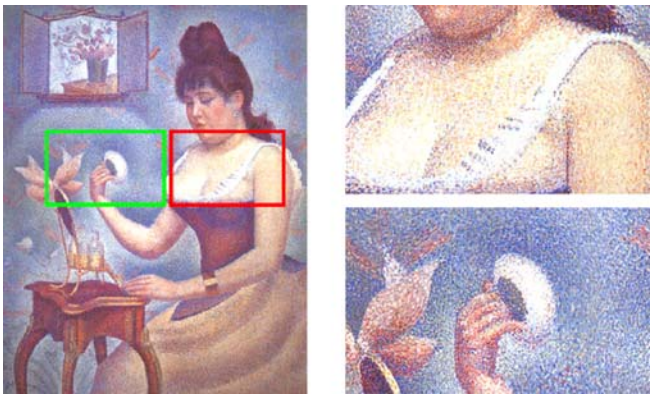

Fig. 2. *Left*: the image of Seurat's *young woman powdering herself*. *Upper-right*: an enlarged portion of the *left image* marked with *red*, where one can see the crease edge between the woman's body and her left arm being strengthened. *Lower-right*: another enlarged portion of the *left image* marked with *green*, where one can see the halo effect around the powder puff on the women's right hand

Seurat's ideas about the complementary colors. We will address this issue in the Sect. 4.

3.4 Divisionism

As mentioned before, pointillism or divisionism, represents a desired color by putting two primitive colors together properly, especially when the observation is at a distance. Figure 1 also demonstrates an example of such.

4 Non-photorealistic rendering of Seurat's pointillism

After careful and detailed examinations of many Seurat's paintings, our task is to simulate these observed features by computers. Before illustrating our simulations step by step, one thing worth mentioning is the color space conversion. Although representing pixel colors in terms of RGB is the most common approach, it is well known that such a representation usually presents high-correlation among the three channels, that is, a large value in the red channel usually also suggests large values in the rest two channels. This also explains why a separation of hue, saturation and lightness is very popular among various image or video processing techniques, as it could provide better visual quality after different kinds of manipulations. Here we select the well-known CIE$L^*a^*b^*$ or CIELAB color system, shown in Fig. 3, as our internal color manipulation format, not only for its decoupling of hue, saturation and lightness, but also for its *perceptually uniformness*, which means that the same amount change of value would result in the same amount change of visual perception. For example, a change of pixel value from 100 to 110 in the CIELAB color system would cause the same impression as the change from 0 to 10, which in general is not true for other color systems. Such uniformness could facilitate our image processing so that we only need to concern how values change would affect the system, without worrying what these values originally are. Gooch et al. [8] adopted a similar approach for converting colored images into grayscale ones.

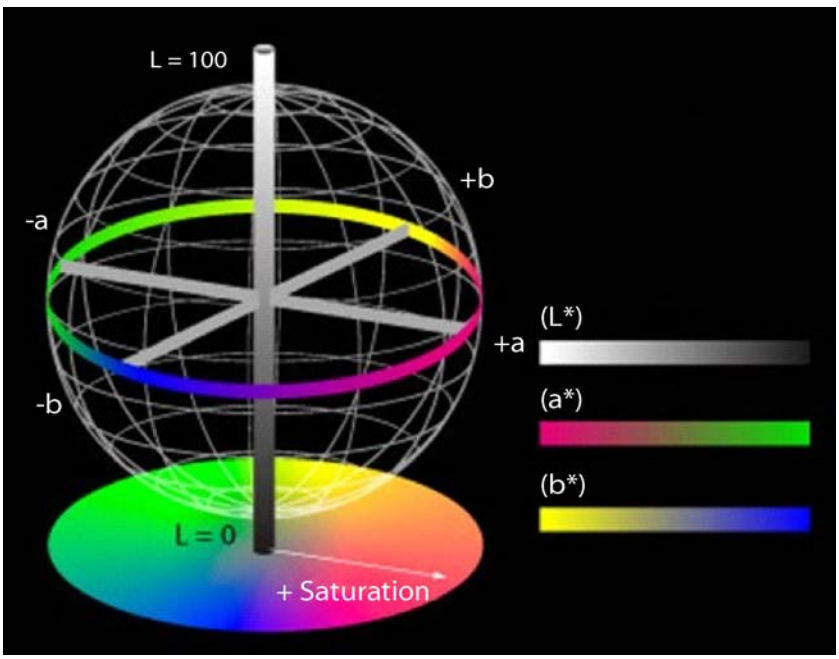

Fig. 3. The CIELAB color system

4.1 Eleven colors

Recall from the previous section, instead of using all the colors in the color gamut, Seurat only made use of a relatively few number of colors during his painting process, and each such chosen color could be *adjusted in its saturation or lightness*. To have a faithful simulation of his painting style, it is reasonable to also enforce such a constraint. However, to our knowledge, there seems to be no formal definition of these eleven colors. To address this, in our current prototype implementation, we make use of the 72 colors in Michael-Engene Chevreul's color circle, shown in Fig. 4, as our *primary colors* for synthesizing all the colors, as many believe that his color theory has been served as the basis for Neoimpressionism painters. It may be interesting to compare Chevreul's color circle with modern HSV color system, as shown in Fig. 5. In fact, they look different, but still share great similarity. The main differences lie on the following. First the orientation on the color arrangement of Chevreul's color circle is almost opposite to that of HSV color space, with the exception on the order of dark blue and light blue. Second, in Chevreul's color circle, the red color seems to enjoy a much wider spectrum than others. Note that, the second difference, together with the exceptional blue-color ordering, may lead to different complementary colors for a given color. We will come to this issue later. Also note that it is also believed that there exist more than one person that might also have affected Seurat. For example, Ogden Nicholas Rood is one of them, as Seurat also owned a copy of Rood's classic book on *Modern Chromatics* [24].

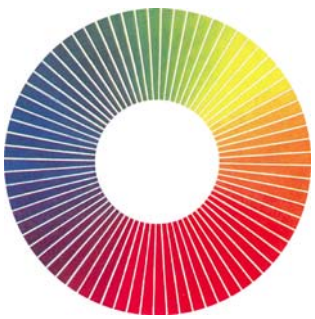

Fig. 4. Chevreul's color circle, originally from [4]. But this electronic image is modified from Fig. 15 in [28]

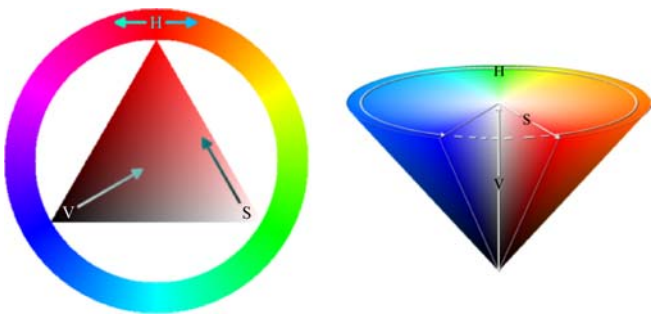

Fig. 5. The HSV color system, cited from Wikipedia. *Left*: a top view of the HSV color space, to be compared with Chevreul's color circle. *Right*: another view which could best explain how hue, saturation, and lightness vary within this space

Note that these 72 colors, derived from [4], are originally in RGB representations, and should be converted to CIELAB ones before being applied in our system. And the conversion formula between RGB and CIELAB is described in the appendix. In fact, in our system, we could also allow users to specify what the available colors are, to obtain maximal flexibility.

4.2 Locations of points

As many classified Seurat's works into pointillism, it is self-evident that how to deal with points is of the utmost importance. Among the many attributes of a point, such as position, color, size, shape and orientation, position is no doubt one of the most important properties of a point. Concretely speaking, each point should be arranged in a proper location in the simulated painting so that the entire set of points as a whole are distributed uniformly with desired spacing among them. To have a simple and efficient solution, we adopt the idea of *Poisson disks* [31] to place points evenly within the entire image, while the pairwise distance between any two points is not less than a pre-specified threshold. The basic idea of Poisson disks is to enumerate all the pixels with a randomized order and a pixel being enumerated could only be retained if it is at least with a desired distance to every previously enumerated and retained pixel. In order to have an unbiased random enumeration order, *linear feedback shift register* or *LFSR* [27] for short, is applied, and the number of involved bits depends on the desired image size to simulate. *For the ease of ensuing discussion, we use* **points** *to denote the locations of the retained* **pixels** *after running the Poisson disk process*. As for the determination of the desired distance among points, more detailed descriptions will be given in Sect. 4.5, for it is closely related to the size of a point. Note here the size of a point in general is larger than a pixel, as we will associate a mask with each such point for simulating the real point strokes used during a painting process.

4.3 Layering structure of points

Currently there are overall four layers in our implementation. The first, or the bottom layer is to serve as the background. Considering that a background color is often used to set the basic tone, we initially perform a *segmentation* process to partition a given image into regions of pixels with similar colors, and then we locate the *brightest* point in each segment for using its color to paint all the points falling within the same region. Our current im-

plementation of the segmentation is a *flooding* algorithm, which starts with the pixel at the upper-left corner of the input image and compares its L^* value (in the CIELAB representation described before) with that of all its neighboring pixels. All pixels whose L^* value differences fall within a threshold are included in the same segment where the initial seed belongs to, and all such included pixels are also served as new seeds for enclosing more pixels. Here the threshold, denoted by *DIFF* hereafter, is set to be *one standard deviation* of the L^* values of all the pixels of a given image, to respect the variations presented from image to image, and this threshold is also *user-tunable*, to seek for more pleasing results. Honestly speaking, to have satisfactory results when faced with a variety of images, the tuning of several parameters seems inevitable. We deem this moderate amount of user involvement acceptable, as our job in its essence is art-specific, thus requiring subjective adjustments from time to time during the entire simulation process. Note that the segmentation is performed on the original image level, i.e., on the pixel level, while the background coloring process is applied only on the point level. In other words, after the image is partitioned into regions, we only color those points within each region. However, coloring only points rather than pixels may leave undesired large blank areas from place to place, and this does not match well with what we observed in Seurat's paintings, where in general point strokes are densely placed throughout the image. As a result, background points, not only are used to set the basic tone, but also are responsible to fill up the image properly to avoid a sparse look. Between the segmentation and coloring process, there is one more step that we may add to filter out areas that are too small to be treated as a single region, so that the resulting image will not be too fragmented. This could be easily achieved by merging a region that is too small in terms of its area, with its neighboring region having the closest color. Such an area threshold will be denoted by *AREA* in the following discussion. Notice that such a merging should not be performed if all its neighboring regions are very dissimilar, and again the similarity metric can be guarded by a threshold to prevent important features from disappearing.

The second layer, sitting on top of the first layer, is to add random but controlled variation, in terms of colors, to the previous points colored as backgrounds. The reason behind this randomness is apparent: it's natural that slight variations exist from stroke to stroke. The variations could manifest themselves in many ways, such as color, size, shape and orientation, and we will discuss each of them in turn. In addition, to mimic the dense coverage observed from normal Seurat's paintings, points of this layer are independently added in by applying the Poisson disk method for another run.

We perform the operation of edge enhancement on the third layer. Through the use of *Canny edge detection* algorithm, we could identify essentially two kinds of edges: silhouette edge and crease edge. The distinction between the two is to be elaborated later. We then apply different edge enhancement schemes to strengthen these edges accordingly. Unlike what we did on the second layer, this enhancement process only replaces the colors of the selected existing points from the second layer without generating any new points.

The fourth layer, or the top layer, is used to deal with complementary colors. As mentioned before, complementary colors, commonly used in Seurat's paintings, serve as contrasts to consequently enhance the original colors. We selectively choose points from the second layer and substitute their complementary colors for their original colors. How to find the complementary color of a given color will be explained shortly. And it could be inferred that the distribution of points in this layer should be sparser than that of the previous layers.

4.4 Colors of points

Following the description of the four-layer structure of points, here comes the most important aspect of a point, i.e., color. We first explain how color perturbation is realized on the second layer of points. The central idea is to perform the modulation in the CIELAB domain, as the same amount of change in values leads to the same amount of alteration in visual perception in this color system. For a given color, represented as (L^*, a^*, b^*) in its CIELAB representation, and marked with P in Fig. 6, we could derive three associated thresholds to limit the range of perturbation, thus forming a 3D region of variation. The first threshold, L_m, represented as the dotted pink line in Fig. 6, normalized to be in the range of 0 to 1, controls the randomization range in the L^* or the lightness channel, under the constraint that the maximal change does not go out of bound. The second threshold, S_m, represented in the same figure as the dotted blue line, governs the randomization range of the saturation of a given point. It is determined by the length of the vector formed by the a^* and b^* components of P, and such a threshold is also nor-

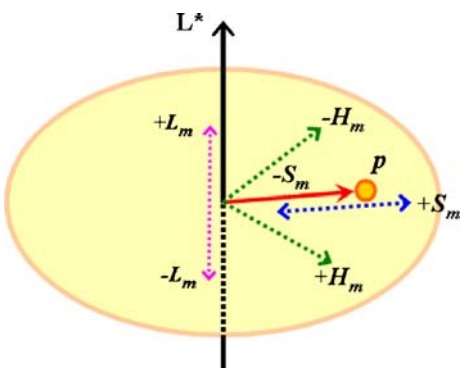

Fig. 6. Color perturbation process in the CIELAB color space

malized to be in the range of 0 to 1 so that it is easier to manage. The third threshold, H_m, represented as the dotted green line in the figure, controls the randomization range of the hue of P. It is decided by the vector (i.e., the direction or angle) formed by the a^* and b^* components of P, and the range is from 0 to 360. Furthermore, such a variation distribution should have its peak centered at the original color ($P = L^*, a^*, b^*$) to be perturbed, therefore we enforce a *3D Gaussian* distribution on top of the 3D variation range. In practice, this 3D Gaussian distribution could be implemented by the *Box–Muller* method [2]. And in our current implementation, the thresholds are set as *three times* of the corresponding standard deviations in the 3D Gaussian distribution so that more than 99.8% of the variation will be contained within the desired range.

However, it is not necessarily true that every color in the CIELAB domain, as well as its perturbed one could be directly representable through points of primary colors. Recall that the basic tenet of pointillism is to juxtapose primary colors appropriately to achieve the impression of a given non-primary color, but theoretically how this is done is still not clear. To address this, we have developed an approach that essentially applies a standard *dithering* of Chevreul's primary colors to produce non-primary colors. More details regarding this approach will be given in Sect. 4.9, where a closely related topic on complementary colors will be discussed altogether in one shot.

4.5 Sizes of points

To have a faithful simulation, we begin our study by making careful observations on Seurat's numerous paintings. In particular, we have to infer the point sizes used in his paintings in the real world, and calculate the corresponding point sizes on the image to be simulated and displayed on our screen at a desired resolution. This inference could be easily done by the following. We first measured the point sizes from the downsized reproductions of Seurat's paintings whose real dimension information is available, and then the point sizes for the real paintings could be derived. To mimic similar points on a screen with a fixed dots per inch (dpi) resolution, the corresponding point sizes could then be determined. In our implementation, the side length of a point is set to be 9 to 17 pixels after the measurement and conversion calculation. For example, from the painting named as *The Harbour Entrance, Honfleur, 1886*, the original painting is known to be 54 cm × 65 cm, and we measure the point sizes from a downscaled version of it on a book with the dimension of 20 cm × 24 cm. The side length of points ranges from 0.1 cm to 0.2 cm; therefore we could conclude that the side length of points in the real paintings ranges from 0.27 cm to 0.54 cm. Now we want to draw such points on our 72 DPI screen, which means a screen that could show 72 dots per inch. As 0.27 cm is roughly 0.11 inch, therefore this length corresponds to 0.11×72, or about 8 pixels. Consequently the side length of points to simulate should be in the range of 8 to 16 pixels. In our implementation, we use 9 to 17 pixels instead for the reason that the central position of a point is easier to define.

For better quality, the point sizes may be different from layer to layer. As mentioned previously, the background layer is used to set the basic tone, as well as to fill up the gaps. Therefore a point size in this layer might be slightly smaller than desired, thus also resulting in a little denser spacing among the points. On the contrary, points on the second layer could have desired sizes as they represent the major tones.

Furthermore, to emulate real paintings, we should not ignore the minute differences from stroke to stroke. To approximate this effect, we must allow the point size to vary accordingly, and this can be achieved by applying another normal distribution peaked at the desired point size, with a properly set standard deviation to control how the distribution should be clustered. Note that, to simplify the ensuing rendering process, we enforce a *cutoff* value to filter out generated sizes that are too big or too small, thus equivalently corresponding to a *windowed normal distribution* to restrict the size variation range.

Once we allow the variation of point size, this generation process should be coupled with the aforementioned Poisson disk approach for assigning the point locations, and then a point could be placed into the image without colliding with others. For this reason, the original algorithm for Poisson disk location generation is slightly modified, so that each time as a new pixel is enumerated by the LFSR randomized order and to be placed as a point into the image, the distance it should check against is now given by a windowed normal distribution. If the generated distance is d, then this new point location must be at least d pixels away from all previously enumerated and retained points. However, undesired effects may be resulted due to the enumeration order. For instance, assuming pixel x is assigned the distance of 6, and it passes the tests to stay as a point. This indicates that all former points are at least 6 pixels away from pixel x. As the process proceeds, at some point in time, it is quite possible that, another pixel, say y, whose accompanying distance is 4, is to be placed near pixel x. From pixel y's point of view, it could be kept as a point as long as all prior points are at least 4 pixels away. However, from pixel x's point of view, y's existence now violates its original intent to be at least 6 pixels away from others. To solve this problem, as a pixel is to be retained as a point, it should also record together with itself the distance that was used for testing. After this modification, now from y's point of view, every prior point z has to be at least d pixels away from pixel y, where d is the larger one of 4 and e, with e being the distance value stored with point z.

In our current implementation, in the background layer, the distance between points is 3 pixels, while point size is 5 pixels. In the second layer, the distance between points is randomly chosen from 5 to 9 pixels, while the point size is

from 9 to 17 pixels. As mentioned, such a randomization is to simulate the different sizes of strokes observed in normal paintings. The edge enhancement is done selectively on the points deployed in the second layer. For adding complementary colors, we also select points from the second layer but with a larger distance. The distance ranges from 20 to 40 pixels, so that we could make the contrasted colors more prominent without being offset too much.

4.6 Shapes of points

Once the location and size of a point are settled, the next property comes to play is shape. For each point to have a natural look, like what a painter would do with his point strokes, each point should not always bear the same look. A naive approach is to manually scribble a dozen of "predefined" shapes for being the *selection pool*, while each time a shape arbitrarily selected from this pool is randomly perturbed to form a stroke shape on the image. Another way is to start with a solid circle and then randomly and gradually grow from its boundary to "attach" more pixels. Currently we adopt the naive approach for its simplicity and acceptable results, and the shape selection pool is shown in Fig. 7. Alternatively we could try to sample the point shapes from Seurat's paintings for serving as the shapes in the selection pool.

As soon as a shape is chosen, the *diameter* or *principle axis* of it should be calculated for the sake of orientation adjustment to be discussed later. Here the *diameter* of a shape refers to the longest distance between any two pixels on the shape, and it could be found in $\mathcal{O}(n \log n)$ time [26], where n is the number of pixels on the shape. As for the *principle axis* of a shape, it refers to the direction along which the pixels on the shape have the strongest correlation. This axis can be found by the technique of *principle component analysis*, or *PCA* for short, which normally involves a higher time complexity than finding the diameter of a shape. In this paper we adopt a brute-force like method which simply tests all the boundary pixels of a shape to find the diameter, and use the associated two pixel positions to approximate the main trend of a shape. This performance overhead is negligible as there are only a fixed number of shapes to deal with at the *preprocessing time*.

4.7 Orientations of points

We first identify the main variation direction of a point shape, and then we could rotate the oriented point toward the direction that is perpendicular to the gradient direction calculated at the central pixel of a point, as the gradient vector always points to the direction along which the value changes most. In practice, the gradient vector of a pixel could be approximated by using the *central difference* formula. Figure 8 shows such a process.

Fig. 7. The stroke shapes used in our current implementation

4.8 Edge enhancement

Edge enhancement for 3D rendering or illustration has been a focus for many researchers [1, 7, 25]; however, the approach of dealing with 2D images is quite different. To simulate the edge enhancement effect regularly presented in Seurat's paintings, we first apply Canny edge detection [3] to identify potential edges. There are basically four steps. First, a *Gaussian filtering* is performed to remove undesired noise. Second, the well-known *Sobel operator* is adopted to calculate the gradient vector for a pixel. Third, the *non-maxima suppression* is applied to only retain those pixels whose gradient vectors have the largest magnitudes around the neighbors along the direction of the gradient vectors, and the resulting image is composed of thin lines, as shown in Fig. 10. Fourth, two selected thresh-

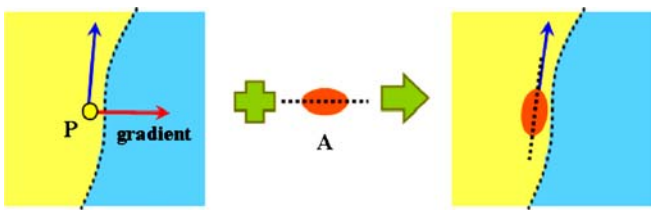

Fig. 8. The process to orient a point. *Left*: we first find the gradient vector associated with a given pixel position P. *Central*: we identify the principle axis of point A to be placed on P. *Right*: we orient point A so that its principle axis is perpendicular to the gradient direction of P

Fig. 9. The source image of a pink flower that is used in Figs. 11, 12 and 14

olds, T_1 and T_2, with $T_1 < T_2$ are used to derive two images, containing pixels whose associated gradient magnitudes are larger than T_1 and T_2, respectively. We will refer to the corresponding images as IM_1 and IM_2 for convenience. Starting with an pixel sitting on one end of an edge in IM_2, we move toward its neighboring pixel, say x, if x appears in IM_1. Such a traversal continues until we have visited all end point pixels in IM_2, and the trace of all such traversal forms the set of edges that we want. Basically it means that the pixels in IM_2 are the candidate edge pixels extracted from the third step, while pixels in IM_1 are used to determine how these pixels could get connected to obtain the final edges. Figure 11 shows one sample resulting image. Based on these preset thresholds and the algorithm, *silhouette edges* and *crease edges* are further distinguished. The former usually represents the edges along the boundary, thus being more salient, whereas the latter mainly refers to the internal edges. In practice, we differentiate an edge's type by its length according to two thresholds: L_1 and L_2. If the length is smaller than L_1, then the edge is simply discarded; if it's larger than L_2, then it is treated as a silhouette edge; otherwise it is thought to be a crease edge. We should point out that making a judgment on whether an edge is a crease edge or not could be very subjective, we therefore leave such a decision to users. However, as sometimes a global setting on L_1 and L_2 may not match with what users desire towards each individual edge, we also allow users to explicitly select an edge and make it a silhouette edge. More sophisticated schemes or user assistance could also be adopted for a more precise or desired classification.

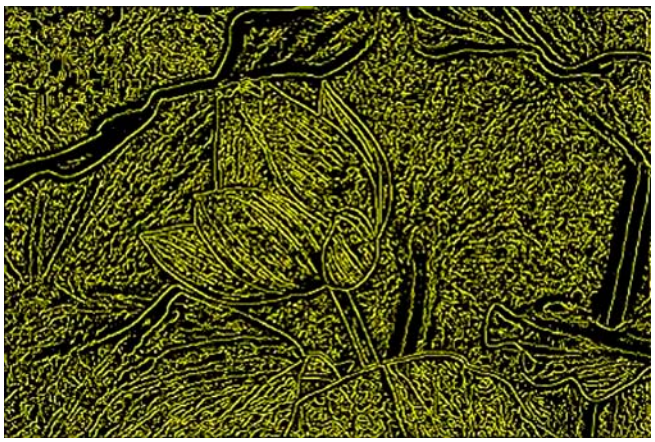

Fig. 10. The resulting image after applying the non-maxima suppression. The source image is shown in Fig. 9

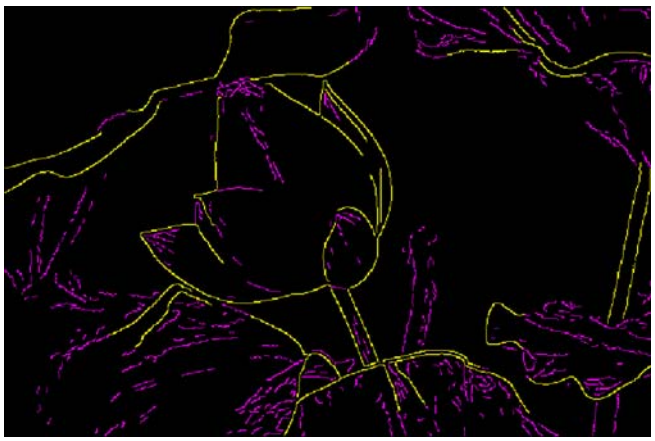

Fig. 11. The resulting image after the edge detection process. The source image is shown in Fig. 9. Here T_1 and T_2 are set to be 0.05 and 0.6, respectively, while L_1 and L_2 are 15 and 50, respectively

4.8.1 Silhouette edges

Once an edge is classed as a silhouette edge, we proceed by strengthening it to generate the so-called *halo effect* mentioned previously. To do this, we first need to determine the affected region, which is in turn determined by an associated radius, in terms of pixels. We could identify such regions by "rolling" a circle with a specific radius along all silhouette edges. Next, we need to further distinguish the *bright side* and *dark side* from an edge, as we will enforce different enhancing policies for them, and the desired result is shown in Fig. 12, where both sides of all recognized silhouette edges are marked, but with different colors. Note that in order to have a clean cut between both sides of an edge, our circle rolling process in fact only needs to be performed on the central part of each silhouette edge. Whether a region is a bright side or dark side can be determined by comparing its average pixel brightness against each other.

To enhance an edge, the basic idea is to enlarge the contrast; or more specifically, to make the bright side brighter and at the same time the dark side darker. We also hope such a contrast enhancement gradually decay as we move farther away from the corresponding edge. This is achieved by applying the following non-linear function:

$$\text{ratio} = \frac{a_{\text{dark}}}{2}\left[2\left(\text{dist} - \frac{1}{2}\right)\right]^{b_{\text{dark}}} + 1 - \frac{a_{\text{dark}}}{2} \tag{1}$$

to the dark side and similarly another function:

$$\text{ratio} = \frac{a_{\text{bright}}}{2}\left[-2\left(\text{dist} - \frac{1}{2}\right)\right]^{b_{\text{bright}}} + 1 + \frac{a_{\text{bright}}}{2} \tag{2}$$

to the bright side, where dist and ratio represent the distance (normalized to be between 0 and 1) from the current point to its corresponding edge, and the adjusting ratio, respectively. This ratio, defined to be the new lightness value divided by the old lightness value, governs how the brightness should be changed for edge enhancement. The four constants, a_{dark}, b_{dark}, a_{bright} and b_{bright}, are used to control the non-linear behavior and the blending with other

Fig. 12. The silhouette edges and their corresponding enhancements for both sides. Source image can be seen in Fig. 9

non-enhanced areas for both the dark side and bright side, respectively. Note that to have a more detailed control, our system even allows the settings of different values of $a_{\text{dark,bright}}$ and $b_{\text{dark,bright}}$ for different silhouette edges. Figure 13 depicts two example non-linear functions used in our current implementations. To be conservative, we also apply a *hard threshold* of *two times of the standard deviation* of the pixel values in this region to guard against the lightness change that may go out of range.

There is, however, one implementation subtlety worth mentioning. It is possible to have two sides belonging to different silhouettes meeting together; and worse yet, one of them is a dark side while the other is a bright side. In fact, even two dark sides or two bright sides contacting each other may still cause problems as both of them may have different enhancement trends, thus creating *discontinuities* at the adjoining edges, as marked with white in Fig. 14. To address this, we identify this kind of edges by re-running the segmentation propagation algorithm to locate pixels striding regions belonging to two different silhouette edges. The aforementioned conflicts could then be resolved for each pixel in the affected region by measuring its distance to both the original edge and the "induced" edge, and taking the smaller one of the two for the ensuing calculation. The corresponding enhancement ratio is shown in Fig. 14 where brighter colors represent stronger enhancements.

4.8.2 Crease edges

The enhancement of crease edges is relatively simple. We do not distinguish between the dark side and bright side of a crease edge, but instead just apply a non-linear contrast enhancement scheme as the following:

$$\text{new}L^* = \frac{1}{2}\left[2\left(\text{old}L^* - \frac{1}{2}\right)\right]^d + \frac{1}{2} \tag{3}$$

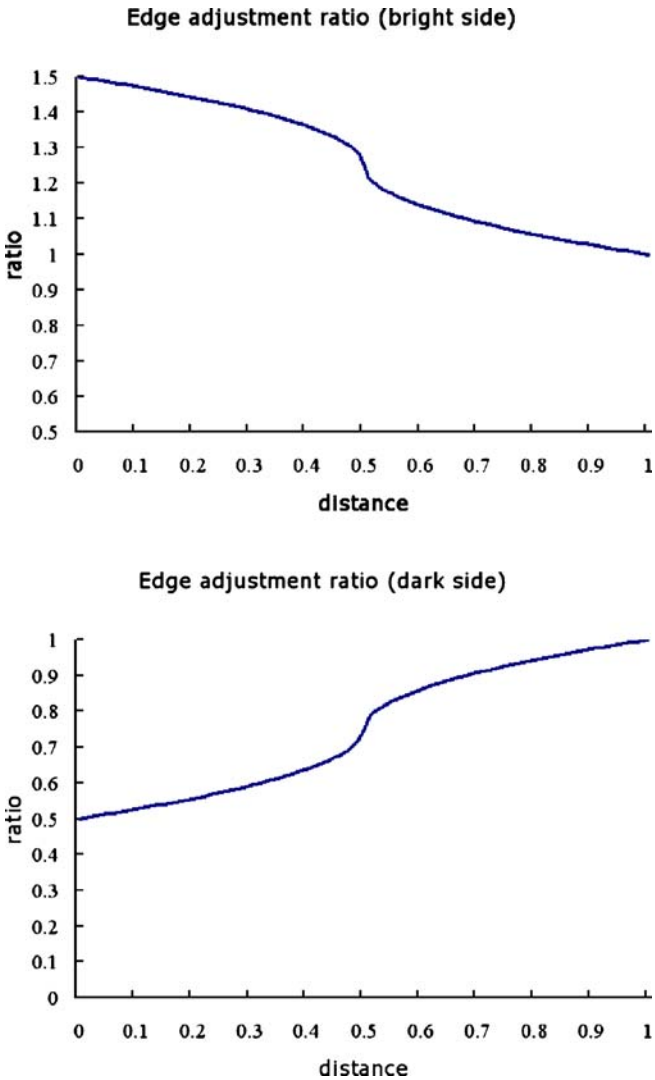

Fig. 13. *Upper*: the non-linear adjusting function for the bright side. *Lower*: the non-linear adjusting function for the dark side. In both cases, all the values of $a_{\text{dark,bright}}$ and $b_{\text{dark,bright}}$ are set to be 0.5

where oldL^* and newL^* represent the original lightness value and adjusted lightness value, respectively. The term d is again used to control the non-linearity, and is set to be 0.5 in our current implementation. The corresponding adjustment function is shown in Fig. 15.

The affected region, compared with that of the silhouette edge, is relatively small. Also note that since the radius of the affected region is small, our enhancement of a pixel in the enhanced region is solely dependent on its value, regardless of how far it is to the corresponding crease edge.

4.9 Complementary colors and color dithering

As mentioned in the previous section, the existence of complementary colors is to enhance the original colors by

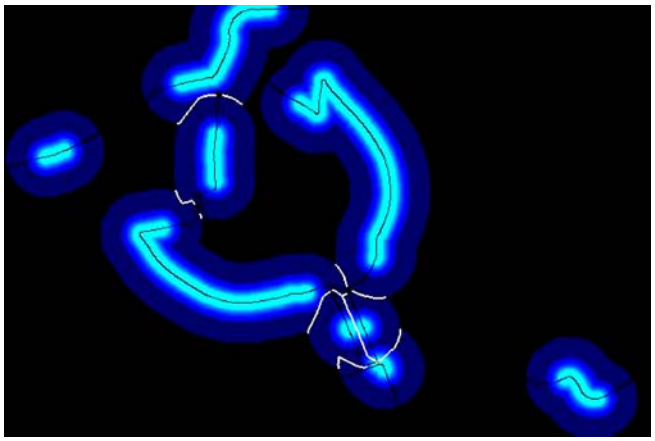

Fig. 14. The non-linear enhancement of silhouette edges. Source image can be seen in Fig. 9

contrast, thus being very important for numerous painting styles. As mentioned previously, it is believed that Seurat was affected greatly by Chevreul's color theory. In fact, Chevreul was the first one to discover the color juxtaposition theory, termed originally as *simultaneous contrast* in his ground-breaking book [4], and basically what Chevreul found is the following rule: if two color areas are to be observed quite close, no matter in space or in time, then each of the color will shift its hue and lightness. For example, if a dark red and a light yellow are put together side by side (or one after the other), then the red will shift as if it is mixed with the complementary color of light yellow (i.e., dark blue), while the yellow will shift as if it is mixed with the complementary color of the dark red (i.e., light cyan). It could be inferred that when two complementary colors are put together, their mutual shifts will be maximal, resulting in the largest contrast.

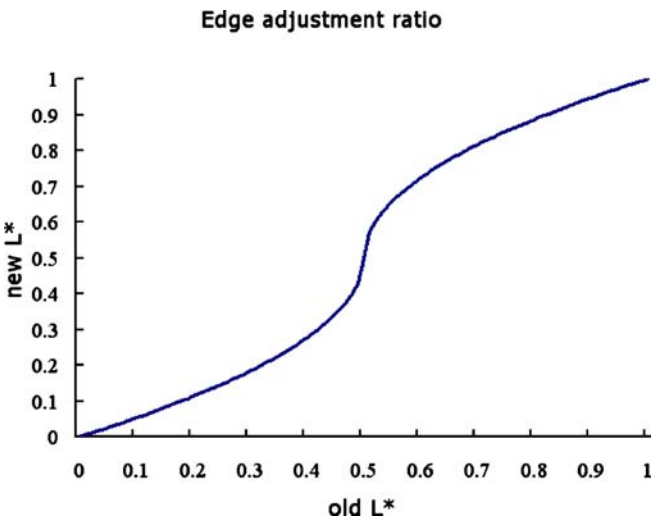

Fig. 15. The non-linear adjustment function of crease edges

One might surmise that finding the complementary color of a given color is trivial; however, we must be aware of the aforementioned differences between Chevreul's color circle, a potential reference palette for Seurat, and the modern HSV color system. Nevertheless, as these two color systems still share much similarity, and at the same time we are not sure of the complementarity mechanism that Seurat employed, we therefore make use of the HSV color scheme for finding the complementary color of a given color. Because of this, we should also make it clear that from now on, the term *complementary color* is still referring to the concept used in modern color theory, that is, a given color and its complementary color are sitting on the opposite sides of the chromatic circle.

Note that Chevreul's color circle, as mentioned in Sect. 4.1, is the fundamental color model that we use throughout for final color representation. Therefore a color, regardless of whether it is RGB or CIELAB, is converted into a representation by Chevreul's color circle before being put onto the simulated image. Such a conversion is done by first matching the color against all 72 color ranges, in terms of of their *hue* values in the HSV model. As for the conversions among RGB, CIELAB, and HSV, they are described in detail in the appendix. If a given color falls within one of the 72 ranges, the color will be represented as it is. Otherwise, if the color is sitting in between two color ranges, then we measure the angular difference to determine the corresponding weights to be used to mix the two neighboring colors for obtaining the desired color. To find the complementary color of a given color, we first find its complementary color in the HSV color space, and then how to represent this complementary color is the same as just described previously.

However, as the gist of pointillism is not through color mixture but via color juxtaposition, we thereby borrow the old techniques of *half-toning* or *dithering* for simulating this effect. Figure 16 demonstrates an example of such, where we assume that color C is to be approximated by the primary 72 colors. After converting it to the HSV space, and according to the hue angle, we could find the nearest two surrounding hues from the 72 colors, say $H_{\min}$ and $H_{\max}$, and the corresponding two weighting factors. Assuming both factors are 0.5, then instead of mixing the color before applying it, we approximate the original color by choosing the most appropriate 2×2 *dithering pattern* to juxtapose the two neighboring colors accordingly. In this example, 2 : 2 is the best one; therefore, we will place one color in two of the four regions while the other color in the rest two, as shown in the lower-left corner in Fig. 16. The color juxtaposition effect could be demonstrated by reducing the region sizes, as if we are looking at the image with a farther and farther distance, and the desired color, i.e., purple, will become apparent, as shown in Fig. 17. Note that in our system, each of the four regions in the

2×2 pattern in fact represents the size of a point, that is, the side length of each region is from 9 to 17 pixels. As a result, the effective resolution is reduced, and a point now becomes a *macro point* compared with the original definition of a point due to dithering. We therefore intentionally scale up the original image to counteract this effect. When applying such a dithering scheme, one might argue that a 2×2 dithering pattern of color juxtaposition may not be fine enough to generate the "impression" of an arbitrary color. However, as an image of pointillism is often observed at a distance, it is quite common that points are relatively small in this sense. Furthermore, we use 72 colors for approximation, together with the dithering pattern for supporting 5 different ratios between the two composing colors, therefore in practice such a mechanism could lead to satisfactory results. We admit that using such a dithering pattern may restrict the *best distance* from which the simulated painting is to be observed. Therefore we have also tried to use 3×3 or 4×4 dithering patterns, but it ends up in the following situation. On one hand, as mentioned, the resulting images have to be further

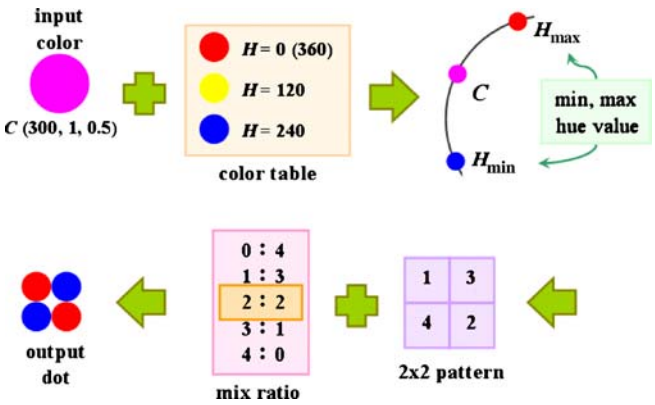

Fig. 16. The color juxtaposition process

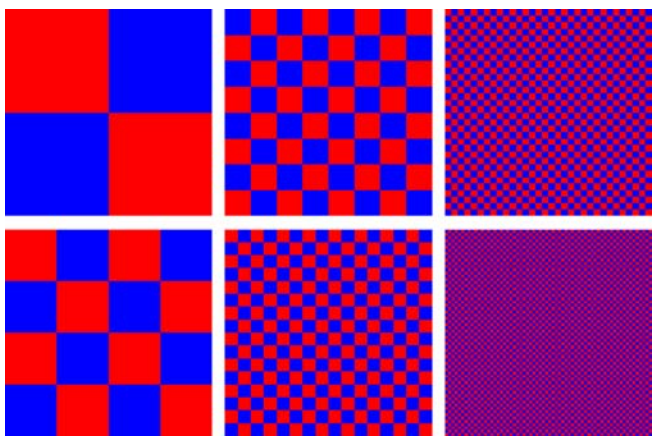

Fig. 17. A color dithering example

scaled up, thus making them even more incomparable with the available simulated images by others which usually are not stored at high resolutions. On the other hand, applying these more complicate dithering patterns does not seem to further improve the results that we have already achieved.

It is also worth mentioning that complementary colors should appear with a relatively lower frequency so that they will not offset the original tones. This could be achieved by selectively replacing a color in the second layer by its complementary counterpart, together with the check to make sure that such a color inversion will happen uniformly within an image.

5 Results

We have conducted our experiments on a Pentium IV 2.8 GHz machine with 512 MBytes memory, running on a Windows XP operating system. Note that to make the paper more compact, we have chosen to shrink all resulting images. However, this should not damage the image quality as each image is associated with a high resolution. Therefore any reader who is interested in seeing more details from an image should always enlarge the corresponding portion.

Before demonstrating and evaluating our rendering results, let us first recapitulate all the related parameters required from the users, so that these results are reproducible. Currently there are totally 14 parameters, and four of them, i.e., *DIFF*, L_m, S_m and H_m could be set automatically. Out of the remaining ten, at least four of them could be set as constants across all the results, thus leaving at most six of them to be provided by users. To facilitate the parameter tuning process, we have also designed a graphical interface that allows us to tune these parameters interactively. The first two parameters, *DIFF* and *AREA*, are used during the segmentation process. *DIFF* represents the segmentation threshold, in other words, neighboring pixels whose lightness difference is less than such a threshold will be grouped together. As mentioned previously, the default value of *DIFF* is set to be the standard deviation of all the lightness values in the input image, and this value is also user-adjustable. *AREA* represents the area merging threshold, that is, a segmentation region whose area is smaller than *AREA* will be merged into its neighboring region that has the least difference in lightness. More details are presented in Sect. 4.3. The next three are L_m, S_m, and H_m, which are used to control the color perturbation range associated with the lightness, saturation, and hue, respectively, as explained in Sect. 4.4. And their default values are set to be three times of their corresponding standard deviations of the input image. The two parameters, T_1 and T_2, are used in the Canny edge detection process, which is described in depth in Sect. 4.8. Parameters L_1 and L_2, are also involved in the edge detection process, where an edge's length equal to or larger than L_2

is considered a silhouette edge; otherwise if it is smaller than L_2 but larger than L_1, then it is classified as a crease edge. To control the non-linear enhancement for both the sides of an edge, a_{dark}, b_{dark}, a_{bright} and b_{bright} are defined accordingly, and detailed equations are presented in Sect. 4.8.1. Similarly in Sect. 4.8.2 we explain how the last parameter, d, non-linearly enhances both sides of a crease edge.

In addition to those values that could be automatically derived from the input image, in our current experiments, the following parameters are constants throughout all the images: $AREA = 500$, $b_{\text{dark}} = b_{\text{bright}} = 0.8$, and $d = 0.5$. And in fact, the value of a_{dark} could be set to 0.5 in all cases, except in the case of Fig. 22, where a smaller value should be used to avoid some undesired side-effect to be mentioned later. Furthermore, the value of T_2 could be set to 0.6 throughout, with the exception that the setting of 0.7 could generate an even better result for the case of Fig. 24. Therefore, only the remaining four parameters, i.e., T_1, L_1, L_2, and a_{bright}, require careful tuning.

We first use the image featuring a *pink flower*, obtained from Hays et al.s' project web page, to demonstrate the pointillistic rendering process, as shown in Fig. 18. All involved thresholds or parameters are described in Fig. 21.

To show that our rendering results are indeed better than those by others, we compare our results with the ones generated by Adobe Photoshop, Hertzmann et al. [14], Hays et al. [12], Jing et al. [16], and Luong et al. [20], respectively. In general, Photoshop could perturb and preserve colors in a reasonable manner; however, when the resulting images are enlarged, the main drawback becomes evident. That is, such images tend to leave too many blanked area, thus failing to properly reflect the original content or simulate an authentic painting style.

Figures 19 and 20 compare the rendering results of ours with the ones generated by Adobe Photoshop and Hertzmann et al.'s work [14], and the source images are downloaded directly from related project web page. In Fig. 19, we could see that the results by Photoshop, as mentioned previously, are not very good. In the enlarged portion, we could observe that the results of Hertzmann et al. deviate too much, in terms of color, from the input image. Our results, on the other hand, not only show a better painting simulation of the original image, but also make some edge strengthening (e.g., the arm, pillow and wrist) and add complementary colors (e.g., numerous blue dots over the dark background) to seek better resemblance to Seurat's painting style. In Fig. 20, again we could observe the excessive color vibration on the earthy background in the results of Hertzmann et al., which may be due to the lack of control on the lightness variation. Our

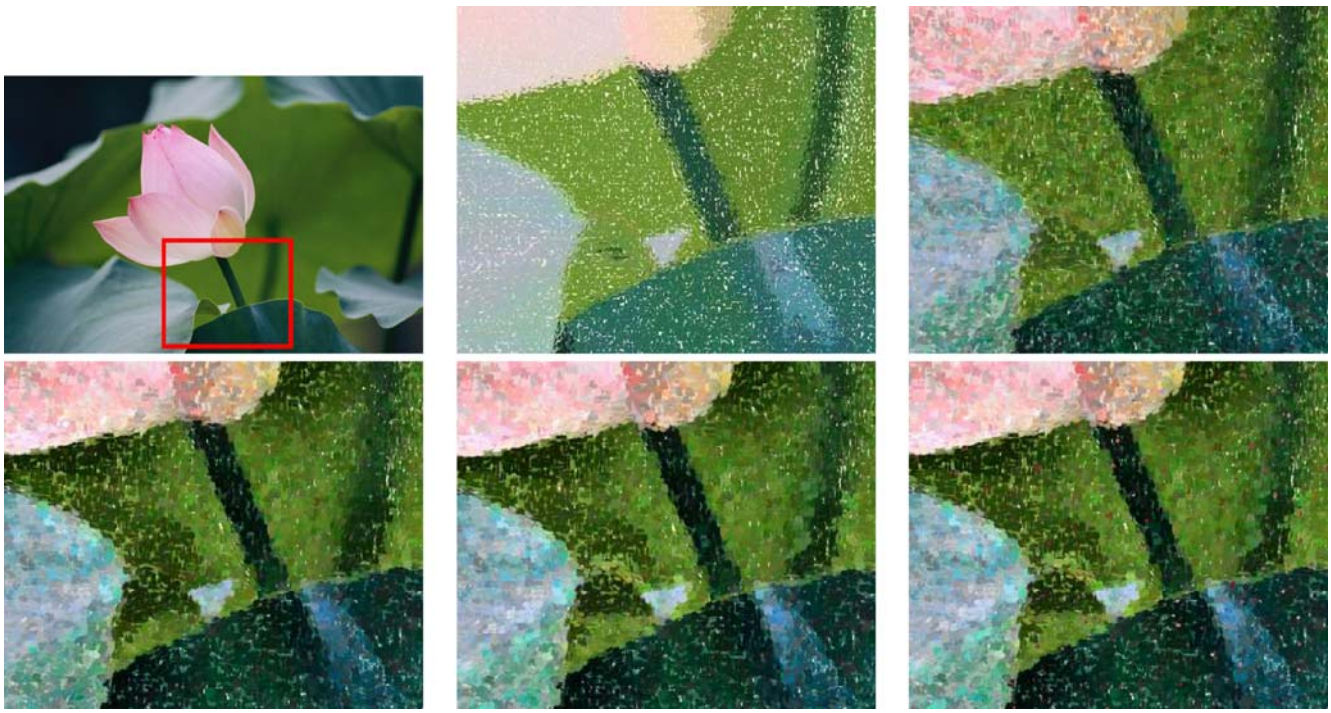

Fig. 18. From *left* to *right* and from *top* to *bottom*. 1. The source image of a pink flower. 2. The corresponding background layer for an enlarged portion marked with *red* in the source image. 3. The previous image added with points of randomized colors. 4. The previous image added with silhouette edge enhancement. 5. The previous image added with crease edge enhancement. 6. The previous image added with complementary colors, thus the final image

Fig. 19. The original image is shown at the *upper-left corner*. The rest of the images is divided into three *columns*, with each *column* demonstrating the results (and their corresponding enlarged portions) from different approaches. Parameter setting or our results: $DIFF = 20$, $L_m = 0.2$, $S_m = 0.05$, $H_m = 45$, $T_1 = 0.15$, $T_2 = 0.6$, $L_1 = 20$, $L_2 = 70$, $a_{\text{dark}} = 0.5$, and $a_{\text{bright}} = 0.2$

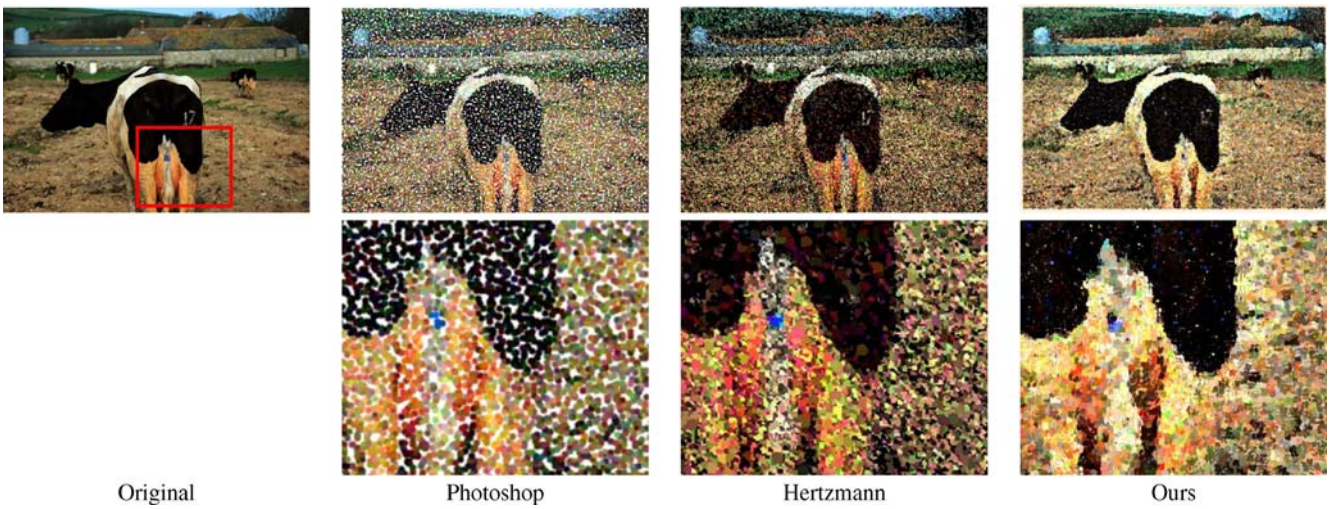

Fig. 20. The original image is shown at the *upper-left corner*. The rest of the images is divided into three *columns*, with each *column* demonstrating the results (and their corresponding enlarged portions) from different approaches. Parameter setting of our results: $DIFF = 21$, $L_m = 0.4$, $S_m = 0.3$, $H_m = 45$, $T_1 = 0.2$, $T_2 = 0.6$, $L_1 = 10$, $L_2 = 50$, $a_{\text{dark}} = 0.5$, and $a_{\text{bright}} = 0.5$

results, on the contrary, deal with colors more properly. Also note the enhanced edges around the cow's contour and the blue dots on the cow's black stripes.

Figures 21 and 22 compare the rendering results of ours with the ones generated by Adobe Photoshop and Hays et al.'s work [12], and the source images are also downloaded directly from their related project web page. In Fig. 21, we could observe huge color variation, which turns some originally green portions into blue ones, and decreases the overall color saturation. Our results, on the other hand, not only preserve the original colors, but also emphasize the flower, the leaves, and the stem by enhancing their corresponding edges. Also note the points with complementary reddish colors for contrasting the original green colors. Care must be taken when dealing with the setting of a_{bright}, which must not be too large as the flower already has very high lightness values. In Fig. 22, the results of Hays et al. show less contrast and a more vague face contour. Instead, our results place proper enhancements, especially around the boundaries of the cheeks, chin, and hair. Care must also be taken when setting the value of a_{dark} for strengthening the lip contour. As the lip is against a bright face, *too much enhancement on its dark side will darken the color of the lip undesirably*.

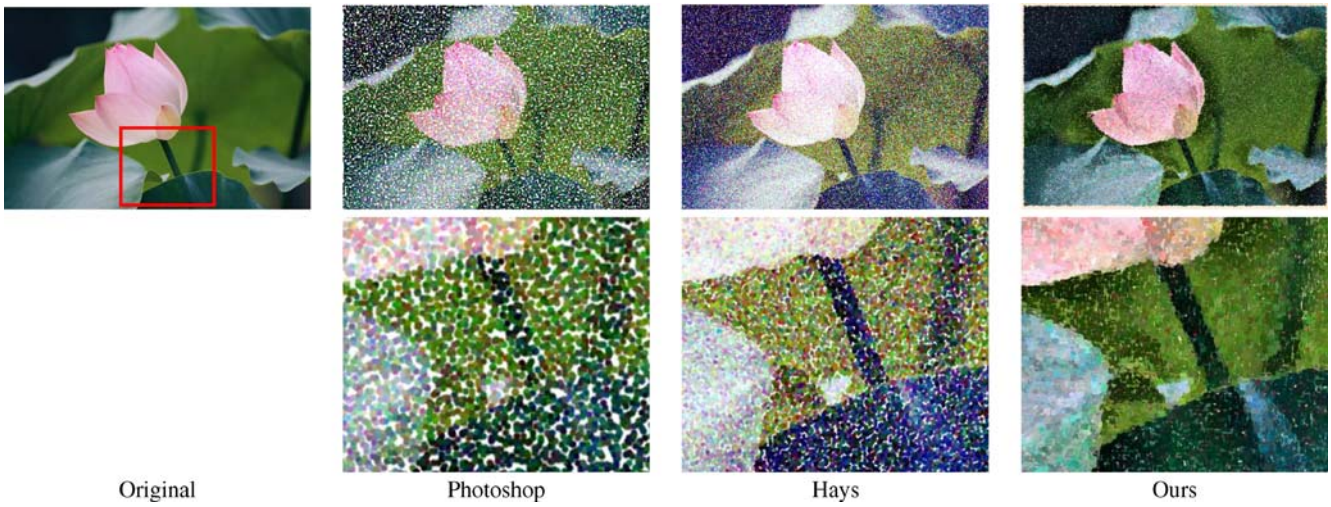

Fig. 21. The original image is shown at the *upper-left corner*. The rest of the images is divided into three *columns*, with each *column* demonstrating the results (and their corresponding enlarged portions) from different approaches. Parameter setting of our results: $DIFF = 15$, $L_m = 0.15$, $S_m = 0.15$, $H_m = 10$, $T_1 = 0.05$, $T_2 = 0.6$, $L_1 = 15$, $L_2 = 50$, $a_{\text{dark}} = 0.5$, and $a_{\text{bright}} = 0.1$

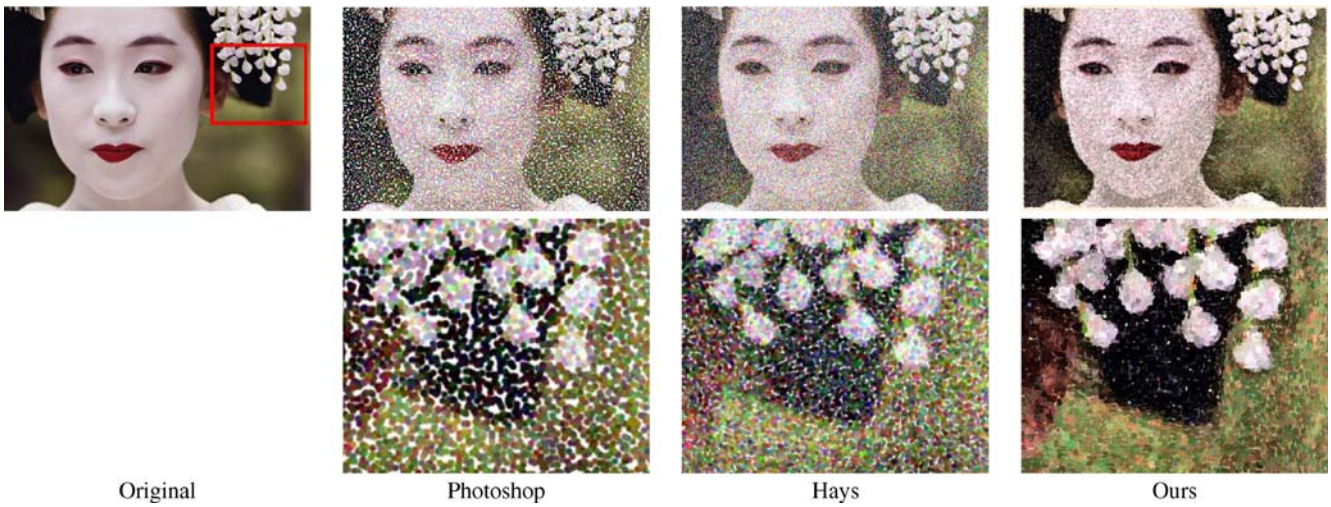

Fig. 22. The original image is shown at the *upper-left corner*. The rest of the images is divided into three *columns*, with each *column* demonstrating the results (and their corresponding enlarged portions) from different approaches. Parameter setting of our results: $DIFF = 20$, $L_m = 0.2$, $S_m = 0.025$, $H_m = 45$, $T_1 = 0.03$, $T_2 = 0.6$, $L_1 = 10$, $L_2 = 50$, $a_{\text{dark}} = 0.35$, and $a_{\text{bright}} = 0.05$

Figure 23 compares the rendering results of ours with the ones generated by Adobe Photoshop and Jing et al.'s work [16], and the source image is downloaded directly from their related project web page. In Fig. 23, we could see that Jing et al.s' results have two apparent drawbacks: the color variation is too large and the strokes are too coarse. Such drawbacks are not presented in our rendered results, and in addition, edge enhancement effects are added to further improve the similarity between our work and Seurat's.

Figure 24 compares the rendering results of ours with the ones generated by Adobe Photoshop and Luong et al.'s work [20], and the source image is downloaded directly from their related project web page. In Fig. 24, we could recognize Luong et al.s' effort to maintain the luminance while perturbing colors, however, the control on hue variation is still not proper. For example, the color perturbation seems to be too large on the central parts of the columns, while at the same time too little on both ends of the columns. Our results not only emphasize the contours of the columns and leaves, but also produce a more vibrant color impression through color juxtaposition, as shown in the partially enlarged portion, which reflects the very spirit of pointillism.

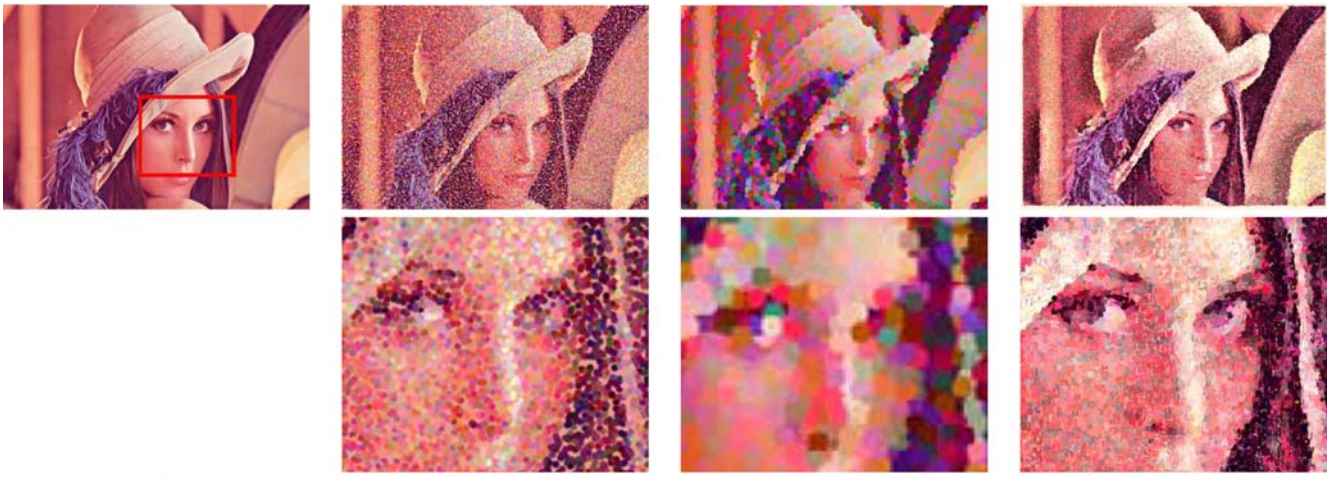

Fig. 23. The original image is shown at the *upper-left corner*. The rest of the images is divided into three *columns*, with each *column* demonstrating the results (and their corresponding enlarged portions) from different approaches. Parameter setting of our results: $DIFF = 14$, $L_m = 0.15$, $S_m = 0.15$, $H_m = 45$, $T_1 = 0.13$, $T_2 = 0.6$, $L_1 = 20$, $L_2 = 100$, $a_{\text{dark}} = 0.5$, and $a_{\text{bright}} = 0.2$

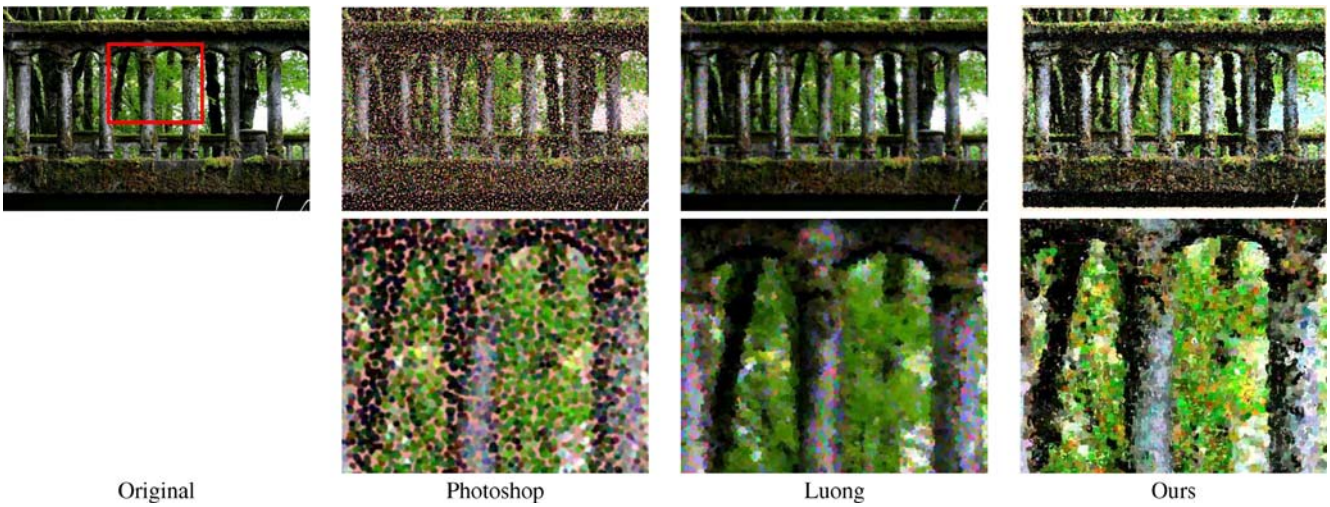

Fig. 24. The original image is shown at the *upper-left corner*. The rest of the images are divided into three *columns*, with each *column* demonstrating the results (and their corresponding enlarged portions) from different approaches. Parameter setting of our results: $DIFF = 26$, $L_m = 0.2$, $S_m = 0.2$, $H_m = 45$, $T_1 = 0.05$, $T_2 = 0.7$, $L_1 = 15$, $L_2 = 70$, $a_{\text{dark}} = 0.5$, and $a_{\text{bright}} = 0.5$

Finally, as Seurat's style is not the main target painting style for many current NPR algorithms; therefore judging our success by simply comparing our rendered results against existing simulation methods is surely not objective. Figure 25 demonstrates our claimed resemblance by placing one of Seurat's paintings, *young woman powdering herself*, and one of our rendered images, as shown in Fig. 23 with previously mentioned parameters, side by side for comparison. The images in the central column compare the color juxtaposition effects, while the right column the halo effects. Through this comparison, we believe we have made a big step towards simulating Seurat's painting style.

As a final remark, we have asked a few students to use our system and give their feedbacks afterwards. Most of them found the system easy to use, but the need of parameter tuning has also made some of them a little bit impatient. As pointed out earlier, usually there are four parameters that require detailed tuning, and on a deeper thought we can see that all of them have something to do with *edge*. The setting of T_1 determines how we want to connect the feature points in IM_2 to form edges (see Sect. 4.8 for details); the values of L_1 and L_2 reflect our decisions on how we distinguish silhouette edges from crease edges; finally the number of a_{bright} represents how we hope the

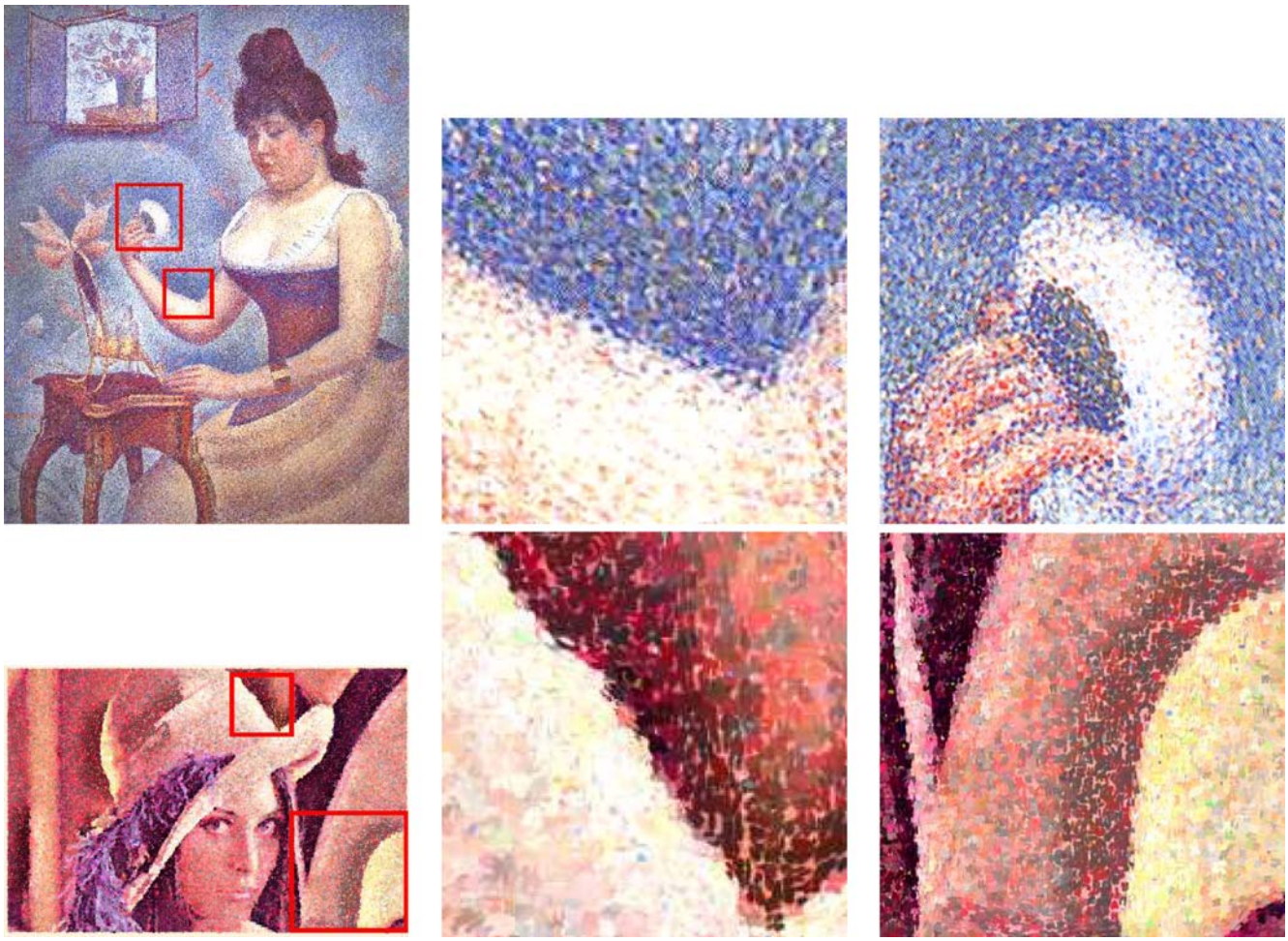

Fig. 25. From *left* to *right* and from *top* to *bottom*. 1. Seurat's painting. 2. An enlarged portion of the painting marked with *red*. 3. Another enlarged portion. 4. A simulated result of our system. 5. An enlarged portion of of our result marked with *red*. 6. Another enlarged portion. Note that the parameter setting of our rendering result is the same as that in Fig. 23

brighter side of a silhouette edge to be enhanced. We must point out that many of these decisions could be very subjective, and image dependent, and thus a universal setting of these values may be impossible. However, as long as we could make our system respond faster and more interactive towards the tuning of these edge-related parameters, we believe that such a system would become even more user-friendly.

6 Conclusions and future work

We have implemented a system that could faithfully perform a non-photorealistic rendering of Seurat pointillism. Our success lies in simulating more closely those features that we commonly observed from Seurat's paintings, such as halos and complementary colors, which have not yet been properly simulated by others. Results are demonstrated and compared with both Seurat's paintings and previous attempted simulations. Honestly, though we regard our work as making a great progress, we believe that there is still room for improvement to bring our simulation even closer to a real painting. For instance, the use of a 2×2 dithering pattern may be too regular and too inflexible; moreover, the dealing with complementary colors may still be different from what Seurat's approach, thus leading to the resulting nuance between the simulated and real ones. In the future, we plan to automate our simulation process by reducing the involved parameter tuning as much as possible, though we know that a complete automation may be impossible. This is due to the fact that sometimes as a high level understanding of an input image might be indispensable, and thus minimal manual adjustment is still required. For example, it is probably fine to allow more color vibration and edge enhancement in Fig. 24 than in Fig. 22, as the latter apparently requires a finer tuning. Nevertheless, a more automatic process will definitely help a user to concentrate more quickly on what

he or she wants to focus, without getting into too many tedious details. Another interesting extension is to apply the proposed techniques to videos, where the dealing of *temporal coherence* will definitely be very challenging. Furthermore, in addition to Seurat's style, we would also like to faithfully simulate the styles of other painters as well. In fact, we believe that some features developed in this work are also applicable to other style. For example, in the observation of some of Van Gogh's paintings, we also found the usage of complementary colors and halos. However, a detailed and extensive survey of a painter's work is always necessary, as what we did for more successfully simulating Seurat's style.

A. Appendix

In this Appendix, we describe the color system conversions that are required in this work. More details could be found in the book *Digital Image Processing* by Pratt [23] or on the *Wikipedia* website.

A.1 Color conversion between RGB and CIELAB

As shown in Fig. 3, in CIE$L^*a^*b^*$ or CIELAB color system, L^* denotes the lightness, or equivalently, the change between white and black, with values ranging from 0 to 100. On the other hand, a^* and b^* denote the hue, where a^* varies between green and magenta, with values ranging from -500 to 500, and b^* varies between blue and yellow, with values ranging from -200 to 200, respectively.

A.1.1 From RGB to CIELAB

We now introduce how to make the conversion from the RGB color system to the CIELAB color system. Note that, before the ensuing conversion takes place, all the pixels' RGB values should be first *normalized* to be within the range of 0 to 1 if they were not originally so. The conversion from RGB to CIELAB requires two passes, that is, from RGB to CIEXYZ, which is a frequently used intermediate color system , and from CIEXYZ to CIELAB, so that the formula is more comprehensible:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}. \quad (4)$$

Then the conversion from CIEXYZ to CIELAB is through the following formula:

$$L^* = \begin{cases} 116 \times \left(\frac{Y}{Y_n}\right)^{\frac{1}{3}} - 16 & \text{if } \frac{Y}{Y_n} > 0.008856 \\ 903.3 \times \left(\frac{Y}{Y_n}\right) & \text{otherwise} \end{cases} \quad (5)$$

$$a^* = 500 \times \left(f\left(\frac{X}{X_n}\right) - f\left(\frac{Y}{Y_n}\right)\right) \quad (6)$$

$$b^* = 200 \times \left(f\left(\frac{Y}{Y_n}\right) - f\left(\frac{Z}{Z_n}\right)\right), \quad (7)$$

where X_n, Y_n and Z_n are the CIEXYZ values of the white point(set to be 0.9515, 1.0000, 1.0886 in this work, respectively), and

$$f(t) = \begin{cases} t^{\frac{1}{3}} & \text{if } t > 0.008856 \\ 7.787 \times t + \frac{16}{116} & \text{otherwise} \end{cases} \quad (8)$$

A.1.2 From CIELAB to RGB

To convert from CIELAB to RGB, we also make two transformations, i.e., from CIELAB to CIEXYZ, and then from CIEXYZ to RGB. For converting from CIELAB to CIEXYZ, we first define the following three terms to ease the description:

$$\begin{aligned} f_y &= \frac{L^* + 16}{116} \\ f_x &= f_y + \frac{a^*}{500} \\ f_z &= f_y - \frac{b^*}{200} \end{aligned} \quad (9)$$

With these terms defined, the conversion goes as the following:

$$\begin{aligned} &\text{if } f_y > 0.008856 \text{ then } Y = Y_n \times {f_y}^3 \\ &\qquad \text{else } Y = \left(\frac{f_y - 16}{116}\right) \times 3 \\ &\qquad\qquad \times 0.008856^2 \times Y_n \\ &\text{if } f_x > 0.008856 \text{ then } X = X_n \times {f_x}^3 \\ &\qquad \text{else } X = \left(\frac{f_x - 16}{116}\right) \times 3 \\ &\qquad\qquad \times 0.008856^2 \times X_n \\ &\text{if } f_z > 0.008856 \text{ then } Z = Z_n \times {f_z}^3 \\ &\qquad \text{else } Z = \left(\frac{f_z - 16}{116}\right) \times 3 \\ &\qquad\qquad \times 0.008856^2 \times Z_n \end{aligned} \quad (10)$$

And then we convert CIEXYZ back to RGB by the following formula:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 3.240479 & -1.537150 & -0.498535 \\ -0.969256 & 1.875992 & 0.041556 \\ 0.055648 & -0.204043 & 1.057311 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}. \quad (11)$$

Finally these $0 \sim 1$ RGB values could be rescaled to the range of $0 \sim 255$ if necessary.

A.2 Color conversion between RGB and HSV

The HSV color system is another popular color system, where HSV stands for *Hue*, *Saturation*, and *Value* (or lightness, equivalently). Hue ranges from 0 to 360, Saturation ranges from 0 to 1, and Value ranges from 0 to 1, respectively. Figure 5 shows the HSV color space.

A.2.1 From RGB to HSV

To convert from RGB to HSV, pixels' RGB values also have to be normalized to be from 0 to 1. And before the conversion, we first define:

$$\begin{aligned} MAX &= \max(R, G, B) \\ MIN &= \min(R, G, B) \end{aligned} \tag{12}$$

And then the conversion is done through the following:

$$H = \begin{cases} 60 \times \frac{G-B}{MAX-MIN} + 0 & \text{if } MAX = R,\ G \geq B \\ 60 \times \frac{G-B}{MAX-MIN} + 360 & \text{if } MAX = R,\ G < B \\ 60 \times \frac{B-R}{MAX-MIN} + 120 & \text{if } MAX = G, \\ 60 \times \frac{R-G}{MAX-MIN} + 240 & \text{if } MAX = B, \end{cases}$$

$$S = \frac{MAX - MIN}{MAX}$$

$$V = MAX$$

$$\begin{aligned} &\text{if } MAX = MIN \text{ then } H \text{ is undefined} \\ &\text{if } MAX = 0 \text{ then } S \text{ is undefined} \end{aligned} \tag{13}$$

A.2.2 From HSV to RGB

To convert from HSV to RGB, we should first check the value S of a pixel. If S is zero, then it means this color is a grayscale color, then we could just set all RGB values of a pixel to be equal to its value of V. When S is not zero, then the conversion applies the following formula:

$$\begin{aligned} H_i &= \left\lfloor \frac{H}{60} \right\rfloor \mod 6 \\ f &= \frac{H}{60} - H_i \\ p &= V(1 - S) \\ q &= V(1 - f \times S) \\ t &= V(1 - (1 - f) \times S) \\ &\text{if } H_i = 0 \text{ then } R = V,\ G = t,\ B = p \\ &\text{if } H_i = 1 \text{ then } R = q,\ G = v,\ B = p \\ &\text{if } H_i = 2 \text{ then } R = p,\ G = t,\ B = t \\ &\text{if } H_i = 3 \text{ then } R = p,\ G = q,\ B = V \\ &\text{if } H_i = 4 \text{ then } R = t,\ G = p,\ B = V \\ &\text{if } H_i = 5 \text{ then } R = V,\ G = p,\ B = q \end{aligned} \tag{14}$$

Finally the RGB values could be adjusted from $0 \sim 1$ to $0 \sim 255$ if necessary.

References

1. Appel, A., Rohlf, F.J., Stein, A.J.: The haloed line effect for hidden line elimination. Comput. Graph. **13**(2), 151–157 (1979)
2. Box, G.E.P., Muller, M.E.: A note on the generation of random normal deviates. Ann. Math. Stat. **29**, 610–611 (1958)
3. Canny, J.: A computational approach to edge detection. IEEE Trans. Pattern Anal. Mach. Intell. **8**(6), 679–698 (1986)
4. Chevreul, M.E.: The Principles of Harmony and Contrast of Colors and Their Applications to the Arts. Reinhold Publishing Corporation, NY (1967)
5. Chu, N.S., Tai, C.: Real-time painting with an expressive virtual chinese brush. IEEE Comput. Graph. Appl. **24**(5), 76–85 (2004)
6. Deussen, O., Hiller, S., Overveld, C.V.: Floating points: a method for computing stipple drawings. Comput. Graph. Forum **19**(3), 41–50 (2000)
7. Dooley, D., Cohen, M.: Automatic illustration of 3D geometric models: lines. In: 1990 Symposium on Interactive 3D Graphics, pp. 77–82 (1990)
8. Gooch, A.A., Olsen, S.C., Tumbln, J., Gooch, B.: Color2Gray: salience-preserving color removal. In: SIGGRAPH '2005, pp. 634–639 (2005)
9. Haeberli, P.: Paint by numbers: abstract image representations. In: SIGGRAPH '90, pp. 207–214 (1990)
10. Haller, M.: Photorealism or/and non-photorealism in augmented reality. In: Proceedings of ACM SIGGRAPH International Conference on Virtual Reality Continuum and its Applicatoins in Industry 2004, pp. 189–196 (2004)
11. Hausner, A.: Pointillist halftoning. In: Computer Graphics and Imaging 2005 (2005)
12. Hays, J., Essa, I.: Image and video based painterly animation. In: NPAR 2004, pp. 113–120 (2004)
13. Healey, C.G., Tateosian, L., Enns, J.T., Remple, M.: Perceptually based brush strokes for nonphotorealistic visualization. ACM Trans. Graph. **23**(1) (2004)
14. Hertzmann, A.: Painterly rendering with curved brush strokes of multiple sizes. In: SIGGRAPH 1998 (1998)
15. Hertzmann, A.: Image Analogies. In: SIGGRAPH 2001, pp. 327–340 (2001)
16. Jing, L., Inoue, K., Urahama, K.: An NPR Technique for pointillistic and mosaic images with impressionist color arrangement. In: International Symposium on Visual Computing 2005, pp. 1–8 (2005)
17. Laerhoven, T.V., Liesenborgs, J., Reeth, F.V.: Real-time watercolor painting on a distributed paper model. In: Proceedings of Computer Graphics International 2004, pp. 640–643 (2004)
18. Lee, J.: Diffusion rendering of black ink paintings using new paper and ink models. Comput. Graph. **25**(2) (2001)
19. Litwinowicz, P.: Processing images and

video for an impressionist effect. In: SIGGRAPH 97, pp. 407–414 (1997)
20. Luong, T., Seth, A., Klein, A., Lawrence, J.: Isoluminant color picking for non-photorealistic rendering. In: Graphics Inteface 2005, pp. 233–240 (2005)
21. Meier, B.J.: Painterly rendering for animation. In: SIGGRAPH 96, pp. 477–484 (1996)
22. Nielson, S., Tai, C.: MoXi: real-time ink dispersion in absorbent paper. ACM Trans. Graph. **24**(3) (2005)
23. Pratt, W.K.: Digital Image Processing, 2nd edn. John Wiley & Sons, New York (1991)
24. Rood, O.N.: Modern Chromatics, with Applications to Art and Industry. Appleton, NY (1879)
25. Rusinkiewicz, S., Burns, M., DeCarlo, D.: Exaggerated shading for depicting shape and detail. In: SIGGRAPH 2006, pp. 1199–1205 (2006)
26. Shamos, M.I.: Computational Geometry. Ph.D. Thesis, Yale University (1978)
27. Sklar, B.: Digital Communications. Prentice-Hall, Englewood Cliffs, NJ (1988)
28. Snider, L.: A lasting impression: french painters revolutionize the art world. The History Teacher **35**(1) (2001)
29. Strothotte, T., Schlechtweg, S.: Non-Photorealistic Computer Graphics, Modeling, Rendering and Animation, 1st edn. Morgan Kaufmann Publishers (2002)
30. Winkenbach, G., Salesin, D.: Computer-generated pen-and-ink illustration. In: SIGGRAPH 1994, pp. 91–100 (1994)
31. Yellot, J., I, J.: Spectral Consequences of photoreceptor sampling in the rhcsus retina. Science **221**, 382–385 (1983)
32. Zhu, Q., Shih, Z.: A perceptually-based painterly rendering framework for synthesizing impressionist paintings. In: International Workshop on Advanced Image Technology 2006 (2006)

CHUAN-KAI YANG received his Ph.D. degree in computer science from Stony Brook University, USA, in 2002, and his M.S. and B.S. degree in computer science and in mathematics from National Taiwan University in 1993 and 1991, respectively. He has been an Assistant Processor of the information management department, National Taiwan University of Science and Technology since 2002. His research interests include computer graphics, scientific visualization, multimedia systems, and computational geometry.

HUI-LIN YANG is currently a Ph.D. student in the department of Information Management at National Taiwan University of Science and Technology. She received her bachelor's degree in information management at Shih Hsin University in Taiwan in 2004, and her master's degree in information management at the National Taiwan University of Science and Technology in 2006, respectively. Her research interest is on computer graphics, with a particular focus on non-photorealistic rendering.