A Stochastic Approach to Simulate Artists Behaviour for Automatic Felt-tipped Stippling

Germán Arroyo, Domingo Martín, and María Victoria Luzón

Abstract— **Nowadays, non-photorealistic rendering is an area in computer graphics that tries to simulate what artists do and the tools they use. Stippling illustrations with felt-tipped colour pen is not a commonly used technique by artists due to its complexity. In this paper we present a new method to simulate stippling illustrations with felt-tipped colour pen from a photograph or an image. This method infers a probability function with an expert system from some rules given by the artist and then simulates the behaviour of the artist when placing the dots on the illustration by means of a stochastic algorithm.**

I. Introduction

Stippling is the technique of drawing using dots, which are composed of pigment in a single colour applied with a pen or a brush, changing the density to obtain different shades. The stippling technique can be altered to use colours. This technique must not be confused with pointillism, which uses small distinct dots of colour to create the impression of a wide selection of other colours and blending. The technique of coloured stippling overlaps the dots to shade the illustration whereas it is not allowed in pointillism.

Several problems happen when an artist tries to stipple an illustration using felt-tip pens. The first problem is that the number of different colours of these pens are very limited. A second problem is that the amount of ink and the porosity of the tip makes expensive to stipple a complete illustration, specially with medium-tip markers. The paper is also a problem because a thin paper cannot admit a large amount of ink, and it breaks; on the other hand, a thick paper spreads out the ink too much, in such a way that the shapes become blurred and undefined. But the most important problem is the large amount of time required to finish this kind of work of art. Only a small error in the decision when placing the dots or an error choosing the proper colours can easily derive into a fatal error that implies to discard all the work.

This kind of illustrations are interesting but they are very hard to be produced. If artists could use this technique efficiently, they would produce images of great beauty maintaining the details and the shade even with large dots. Our intention is to provide a system to produce this kind of illustrations automatically using a computer taking as input a photograph or an image inferring the knowledge of the artist.

There are various previous works about stippling[1], [2] but they use only one ink and do not simulate felt-tip pens. In this paper we present a new algorithm that uses the information of the contrast, the borders and the histogram to define a set of cells with a certain probability. This probability is modified by an expert system that uses rules obtained from artists with different experience. The algorithm uses this probability to render the dots in several iterations, which add details progressively. The algorithm detects when the illustration has finished, stopping automatically.

Germán Arroyo is with the Department of Lenguajes y Sistemas Informáticos, University of Granada, C/ Periodista Daniel Saucedo Aranda, Spain (phone: +34 958 244 344; email: arroyo@ugr.es).

Domingo Martín is with the Department of Lenguajes y Sistemas Informáticos, University of Granada, C/ Periodista Daniel Saucedo Aranda, Spain (phone: +34 958 244 344; email: dmartin@ugr.es).

M. Victoria Luzón is with the Department of Lenguajes y Sistemas Informáticos, University of Granada, C/ Periodista Daniel Saucedo Aranda, Spain (phone: +34 958 244 344; email: luzon@ugr.es).

II. PREVIOUS WORKS

Abstract representation of still images was introduced by Haeberli[3] using image colour gradient and user interactivity for painting. Hertzmann[4] places curved brush strokes of multiple sizes on images for painterly rendering. The technique fills colour by using big strokes in the middle of a region and uses progressively smaller strokes as one approaches the edges of the region. Shiraishi and Yamaguchi[5] improves the performance of above method by approximating the continuous strokes by placement of rectangular strokes discreetly along the edges to create painterly appearance. Santella and DeCarlo[6] uses eye tracking data to get points of focus on images and create painterly rendering with focus information. There are good works for painting terrains[7], [8], but they do not work with general models. All these techniques work well on single images but do not simulate colour stippling.

Most of the related research on stippling is focused on generating dots according to the shading of a photograph, paying almost no attention to the shape of dots or the techniques that artists use[9]. Some methods propose using circles instead of realistic dots[2], [10], [11], [12], [13], [14]. This differs noticeably from the illustrations created by artists because natural dots have a gradient. Other methods focus on distributing the dots correctly along a surface according to the shading[15], [16], [17]. In these cases, the use of Central Voronoy diagrams produce easily recognisable patterns. Renderbots is based on automatons but the results, when they simulate stippling, have the same problems with patterns too[18] and it is not able to produce handmade similar results. None of them uses coloured dots because it is a less usual technique.

III. Overview

First, we will discuss the proposed system. It is based on the scheme presented in Figure 1. As we can see, the algorithm has the following steps:

```
1: The information is obtained from the input image.
2: This information is the input of the expert system.
3: The expert system returns some confidences for three parameters.
4: A matrix with probability is generated from this parameters.
5: The algorithm enters in a loop:
6: loop
7:     The new place and size of the dot is computed.
8:     A new dot is composed with a simplified colour of the region.
9:     The dot is blended with the output image.
10: end loop
```

This algorithm finishes when the stop condition is reached. A discrete function of probability will guide all the process to place the dots, in such a way that the order of the dots placement is determined by a matrix of probabilities.

Once the matrix has been generated, the algorithm for dots placement searches a local maximum in the matrix by using a stochastical algorithm. If the local maximum is not 0, a dot is placed there. In other case, it searches again until a number of iterations is reached, and then the algorithm stops.

Once the dot has been placed, the matrix will be updated to remove a certain amount of probability, then the probability of that place is chosen again decreasing each time a dot is placed there.

In the next sections, a more detailed description of the algorithms is presented.

IV. Extraction of Knowledge

In order to extract some knowledge from the image an analysis of the image and an algorithm to process this knowledge are needed. The filtering process is based on what artists do before they draw a stipple illustration. The process may vary from artist to artist, but in general, they begin marking silhouettes and stippling them. The next step is to stipple on the darkest areas of the image. It is also important to stipple in areas where details can be enhanced and where there are different or relevant elements of the photograph. The background is usually irrelevant, so, the most repeated tone in the photograph is ignored by the artist. Colour stippling is not an exception to this rule.

Therefore, the algorithm to extract all this information is the following:

```
1: Let be $I_c$ the contrasted image.
2: Let be $H$ the histogram of the image.
3: Let be $I_b$ the magnitude of the borders of the image.
```

The problem is that artists usually attach different importance to this information according to the kind of illustration to draw, hence this knowledge is not enough to build the rules that our expert system needs.

Due to the fact that images are very general and complex, we have defined a set of properties from the image to construct an expert system that guides the stippling process:

- The amount of contrast of the image (CTR).
- The saturation of the image (SAT).
- The brightness of the image (BRT).
- The complexity of the image (based on the amount of borders) (CMPL).
- If the image has plain background / or if the image has a complex background (based on the histogram) (PLAINBG).
- The number of different tones in the image (TONES).

These properties are integrated in a new structure. This new structure will be called P_s. This structure is used to define the values of an image and used to form the rules of the expert system.

A rule in our system is defined as:

```
IF <Condition> THEN <Goal>
```

In the conditions, the values of the structure P_s are compared in order to trigger a rule, whereas the goals describe operations or results. These operations alter the input image feed-backing the algorithm and improving the solution. These operations are defined as follows:

- Change contrast (CHCTR): changes the contrast of the image
- Change brightness (CHBR): changes the brightness of the image
- Enhance image (ENH): enhances the image by using the border information
- Change saturation (CHST): changes the saturation of the image

The result of the expert system is three parameters that affect the way the matrix of probabilities is setting. These parameters are:

- α: a value that represents how much the contrast of the input image affects the result
- β: a value that represents how much the borders of the input image affects the result
- γ: a value that represents how much the histogram affects the result

With all this information we have designed a set of tests that have been provided to different artists. We have determined a set of simple rules based on what these artists answered. The results of these tests were different according to the degrees of expertise of the artist, so most of the rules of our system were acquired from the most expertise tests. These rules make up the Subject Matter Expert's knowledge base (rulebase).

So, an example rule of our system would be:

```
IF [Brightness>0.5 AND Contrast<0.2]
   THEN [CHCTR(+0.1), BETA = BETA + 0.1
```

Our inference engine is a modified version of BaseVISor[19] to treat with images. This inference engine processes all the chained rules and returns an output

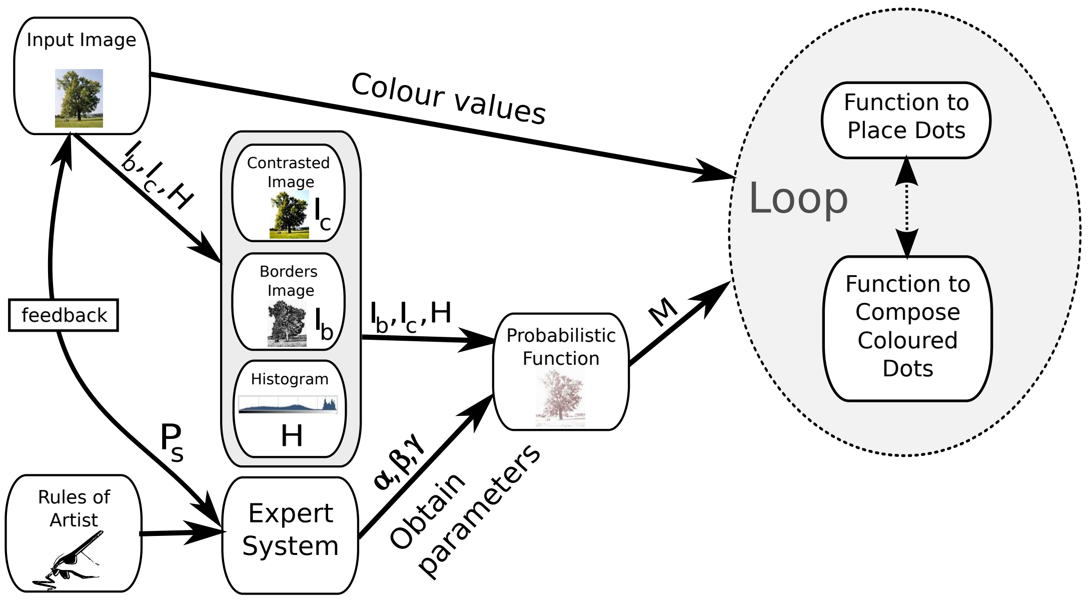

Fig. 1. An overview of the algorithm.

to the system that will be a tuple with tree values: (α, β, γ). This tuple will be the input of the next step of the algorithm to create a matrix with the probabilities.

The matrix of probabilities (M) is a structure that stores, not only probabilities, but also the size of the dot for every position. This matrix has the same size that the previous images.

To generate this matrix, the following algorithm is computed:

```
1: for $x = 0$ to Width($I_c$) do
2:     for $y = 0$ to Height($I_c$) do
3:         $M[x,y] \leftarrow \alpha * I_c[x,y] + \beta * I_b[x,y] + \gamma * H[x,y]$
4:     end for
5: end for
```

Where α, β and γ are the obtained parameters from the expert system.

All the values of the matrix can be normalized easily. The size of the dot can be stored in the matrix in a similar way using the information obtained with these parameters, as shows the next algorithm:

```
1: for $x = 0$ to Width($I_c$) do
2:     for $y = 0$ to Height($I_c$) do
3:         $M[x,y] \xleftarrow{size} \alpha * I_c[x,y] * dpi$
4:     end for
5: end for
```

Where dpi is a variable that adjusts the size of the illustration for printing. Once the matrix has been generated, the next step is iterate until all the dots have been placed.

V. Algorithm for Positioning the Dots

The algorithm for dots placement removes the patterns in the final image. We use a method that is based on a Rejection Sampling algorithm but guided by the matrix of probabilities M.

The algorithm we propose has the following steps for each dot to be placed:

```
1: $iteration \leftarrow 0$
2: $max_p \leftarrow 0$
3: while $max_p = 0$ and $iteration < N\_ITERS$ do
4:     for $i = 0$ to $i < N\_TESTS$ do
5:         $x \leftarrow Random(0, Width - 1)$
6:         $y \leftarrow Random(0, Height - 1)$
7:         $p \leftarrow M[x,y]$
8:         if $p$ is a new maximum then
9:             $max_p \leftarrow p$
10:            $s \xleftarrow{size} M[x,y]$
11:        end if
12:    end for
13:    $iteration \leftarrow iteration + 1$
14: end while
15: if $max_p = 0$ then
16:     End of algorithm
17: end if
18: Generate a dot of size $s$
19: Blend the dot with the final image
20: Update the matrix $M$
```

The algorithm takes as input two parameters: N_ITERS and N_TESTS, and the size of the matrix given by $(Width, Height)$. N_ITERS indicates the maximal number of iterations of the algorithm before deciding that there are no more places to stippling. N_TEST indicates the maximal number of times to search for a higher probability. A preset for N_ITERS = 1.000 and N_TESTS = 10.000 produces good results; these values has been empirically obtained.

The inner loop of the algorithm searches for a local maximum whereas the outer loop skip to another area to search the probability. An iteration of the general algorithm (from line 4 to 13 in the previous algorithm) is shown in

Figure 2. In a first step a series of candidates is randomly generated after a local and a global search. From all these candidates the hightest value is taken. In a second step, a dot is computed and placed in the position given by the best candidate. Finally, in the third step, the probability of the best candidate is decreased in the matrix of probabilities.

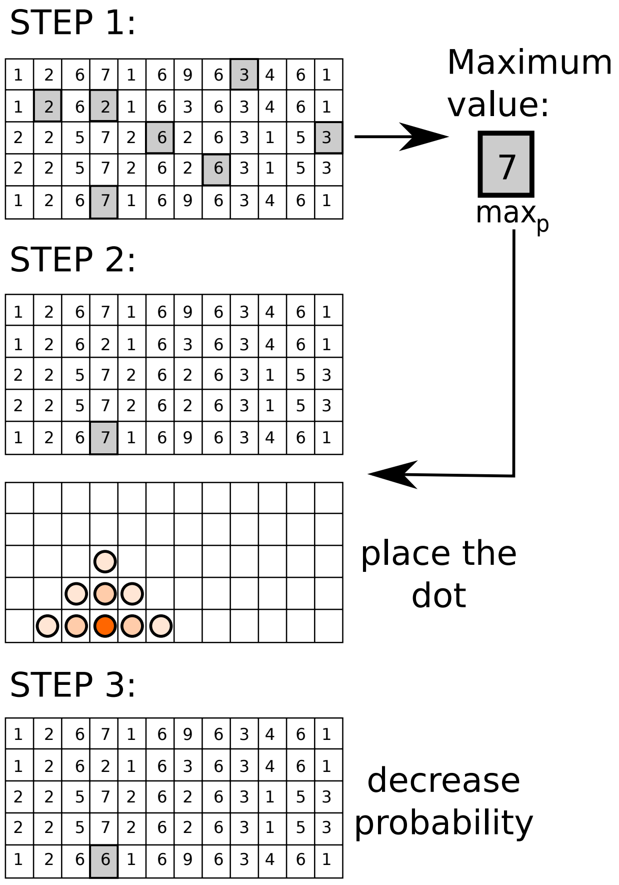

Fig. 2. The general algorithm for an iteration when placing dots.

max_p indicates the maximal probability found, whereas *iteration* indicates the number of iterations before the algorithm decides a local maximum is found.

As we can appreciate, the algorithm would never finish if the values of M do not change. This is the reason for the matrix must be updated. When a dot is placed over the final image, a value is subtracted from matrix M. This value depends of the intensity of the placed dots, therefore dots in the middle subtract a higher value than outer cells. Additionaly, some amount is substracted to neighbour cells of the pixel as shown in Figure 3. The radious of the affected neighbourhood is determined by the contrast in the cell. This value can be adjusted depending on the distance from the centre of the dot. This substraction prevents too many dots overlap in the same area of the image. The values of this substraction can be easily adjusted empirically.

VI. Composing the colour dots

Felt-tipped stippling dots have usually a known size, therefore there is no problem if they are obtained by scanning

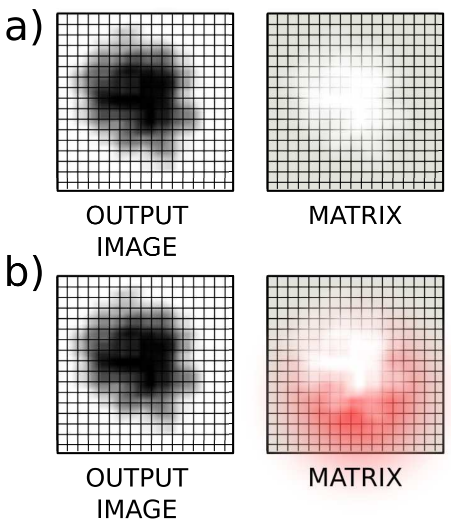

Fig. 3. a) The dot is placed on the output image, and the matrix is updated (in white); b) The neighbour cells are also updated (in red).

a real set of them. Therefore our algorithm reads the dots from a scanned database. This database stores the dots as grayscale independent images. We have used a set of 300 scanned dots with different sizes and resolutions for our database.

When the algorithm decides the position of the dot, its colour is computed by looking at the original image and using the following formula:

$$Colour_{finalDot} = Colour_{image} \cdot GrayValue_{databaseDot} \qquad (1)$$

This formula modifies the intensity of the dot according to the shape of the scanned dot. The alpha value of the image is taken directly from the values of the dot in the database. $Colour_{finalDot}$ is the final colour of the pixel in the dot to be placed. $Colour_{image}$ is the colour of the pixel in the original image, and $GrayValue_{databaseDot}$ is the value of gray of the pixel in the scanned dot.

The dots can be composed in the output image using the following formula for every pixel of the dot:

$$C_d = (1-A)\cdot C_d + (A)\cdot C_s \qquad (2)$$

Where C_d is the colour destiny, C_s is the colour source, and A is the value of the intensity of the dot.

In colour stippling the problem of having only one ink when printing is solved due to the technical capacities of the modern printers, specially if a dye-sublimation printer is used to print the illustration. With this kind of printers the colours use heat to transfer dye onto the paper. The advantage of dye-sublimation printing is the fact that it produces a continuous-tone, where each dot can be any colour. However, today's inkjet printers also produce extremely high quality printings. The reason is that they uses microscopic droplets and supplementary ink colours, producing sometimes even superior colour fidelity to dye-sublimation.

VII. Results

The system is able to generate automatic stippling without user intervention. We have developed and tested them in a software made in C++, the inference engine works as a plugin of the program as a script in Java. These algorithms produces a stippling illustration with felt-tip pens that degrade the colours in an aesthetic way, as shown Figure 4 and Figure 5. The algorithm for dots placement and generation of dots are stable even with complex images as shown in Figure 6, this figure is prepared for printing. The expert system is able to adjust the contrast, the saturation or the brightness of the input image to avoid that plain areas of the image produces noise, hence the result is better in colour and regularity, as shown in Figure 7. It can be appreciated (Figure 8) how differents executions of the algorithms does not produce the same result due to the randomness of the method, although the method is stable and the representation of the figure is the same for the eye.

VIII. Conclusions and Future Work

We have presented an algorithm to automatically draw coloured illustrations and simulates felt-tip stippling. We have introduced an expert system that automatic controls how the algorithms place the dots. We have developed a fast and direct equilibration technique that is based on a probabilistic model. Our system provides a high-quality output with no interaction of the user.

Future research is proposed into how more artistic knowledge can be included automatically within the application. It would also be interesting to be able to generate resolution independent illustrations.

Acknowledgment

Thanks to Vicente Villamón for the photograph of the Figure 4, thanks to Eustaquio Santimano for the photograph of the Figure 6, thanks to Seattle Miles for the photograph of the Figure 5, thanks to Keven Law 8, thanks to Elena Piñar for the illustration of the Figure 7. We have the permission of all the authors to display all the pictures and images.

References

[1] R. Maciejewski, T. Isenberg, W. M. Andrews, D. S. Ebert, and M. C. Sousa, "Aesthetics of hand-drawn vs. computer-generated stippling," in *Proceedings of Computational Aesthetics in Graphics*, no. 3, 2007, p. unknown.

[2] D. Mould, "Stipple placement using distance in a weighted graph," in *Proceedings of Computational Aesthetics in Graphics*, no. 3, 2007, p. unknown.

[3] P. Haeberly, "Paint by numbers: abstract image representations," in *SIGGRAPH'90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, 1990, pp. 207–214.

[4] A. Hertzmann, "Painterly rendering with curved brush strokes of multiple sizes," in *Computer Graphics (Annual Conferences)*, no. 32, 1998, pp. 453–460.

[5] M. Shiraishi and Y. Yamaguchi, "An algorithm for automatic painterly rendering based on local source image approximation," in *NPAR 00: Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, 2000, pp. 53–58.

[6] A. Santella and D. DeCarlo, "Abstracted painterly renderings using eye-tracking data," in *NPAR 02: Proceedings of the 2nd symposium on Non-photorealistic animation and rendering*, 2002, pp. 53–58.

[7] L. Coconu, O. Deussen, and H. C. Hege, "Real-time pen-and-ink illustration of landscapes," in *NPAR 06: Proceedings of the 2nd symposium on Non-photorealistic animation and rendering*, 2006, pp. 27–36.

[8] S. Bhattacharjee and P. J. Narayanan, "Real-time painterly rendering of terrains," in *Proceedings of Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, 2008.

[9] A. Secord, W. Heidrich, and L. Streit, "Fast primitive distribution for illustration," in *Thirteenth Eurographics Workshop on Rendering*, 2002, pp. 215–226.

[10] B. Gooch and A. Gooch, *Non-photorealistic Rendering*. A. K. Peters, July 2001.

[11] X. Yuan, M. X. Nguyen, N. Zhang, and B. Chen, "Stippling and silhouettes rendering in geometry-image space," in *Proceedings of Eurographics Symposium on Rendering*, pp. 193–200.

[12] A. Lu, C. Morris, J. Taylor, D. Ebert, P. Rheingans, C. Hansen, and M. Hartner, "Illustrative interactive stipple rendering," in *IEEE Transactions on Visualization and Computer Graphics*, vol. 9, no. 2, 2003, pp. 127–138.

[13] T. Isenberg, P. Neumann, S. Carpendale, M. C. Sousa, and J. A. Jorge, "Non-photorealistic rendering in context: An observational study," *Proceedings of the Fourth International Symposium on Non-Photorealistic Animation and Rendering, NPAR 2006*, pp. 115–126, June 2006.

[14] S. Kim, R. Maciejewski, T. Isenberg, W. M. Andrews, W. Chen, M. C. Sousa, , and D. S. Ebert, "Stippling by example," *Proceedings of the Seventh International Symposium on Non-Photorealistic Animation and Rendering; NPAR 2009*, pp. 41–40, August 2009.

[15] A. Secord, "Weighted voronoi stippling," in *Proceedings of NPAR*. ACM Press, 2002, pp. 37–43.

[16] O. M. Pastor, B. Freudenberg, and T. Strohthotte, "Real-time animated stippling," in *Proceedings of NPAR 2004*, vol. 23, no. 4, 2004, pp. 62–68.

[17] D. Kim, M. Son, Y. Lee, H. Kang, and S. Lee, "Feature-guided image stippling," *Computer Graphics Forum*, vol. 27, no. 4, pp. 1209–1216, 2008.

[18] S. Schlechtweg, T. Germer, and T. Strothotte, "Renderbots: Multi agent systems for direct image generation," *Computer Graphics Forum*, vol. 24, pp. 283–290, 2005.

[19] C. Matheus, M. Kokar, and R. Dionne, "A demonstration of formal policy reasoning using an extended version of basevisor," in *Proceedings of the IEEE Workshop on Policies for Distributed Systems and Networks, POLICY 2008*, June 2008.

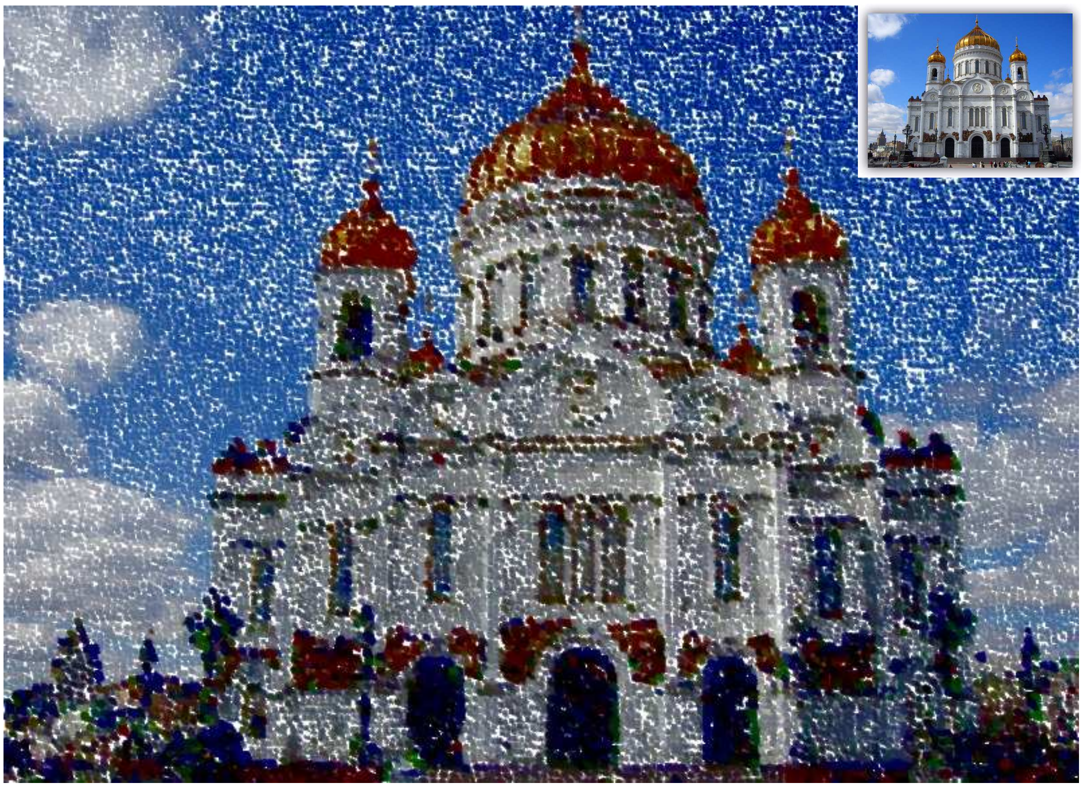

Fig. 4. The result of our algorithm.

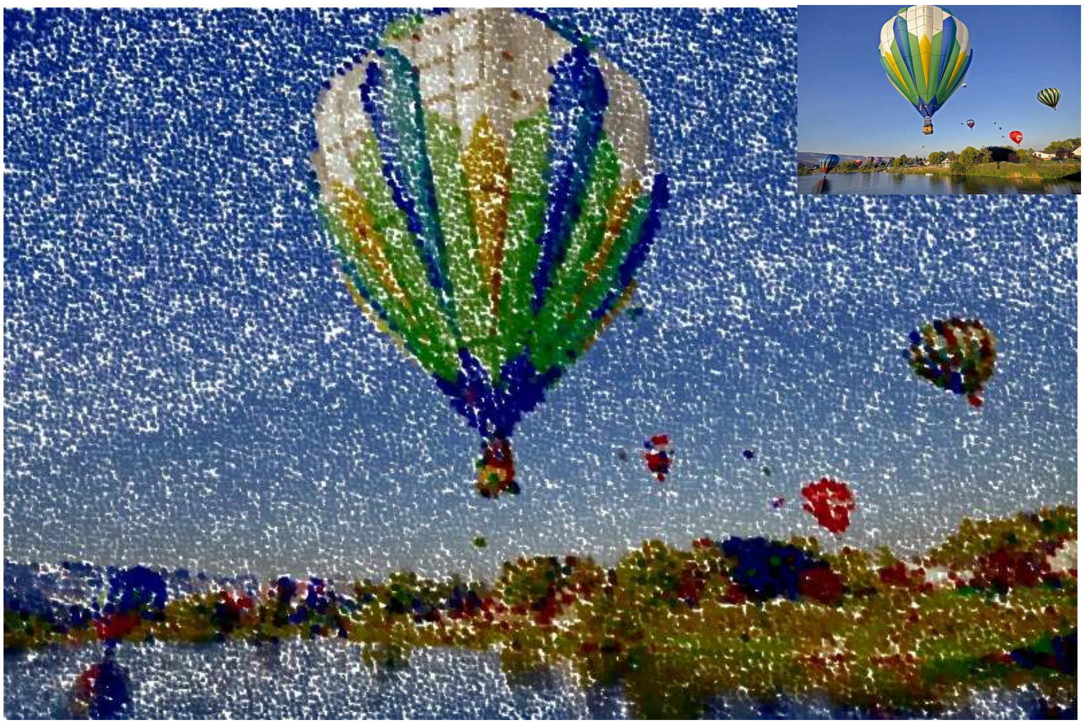

Fig. 5. The algorithm works better with colourful images.

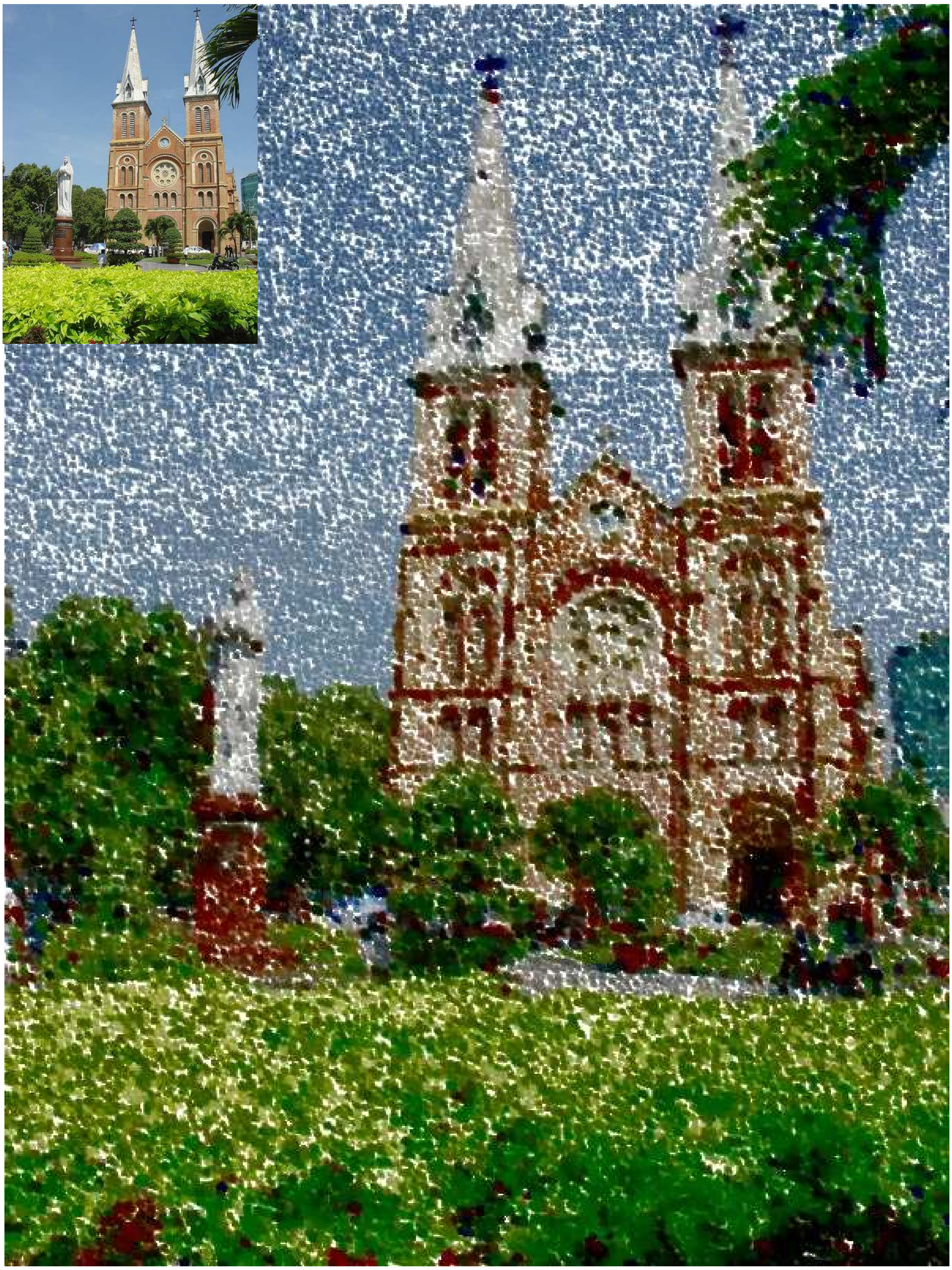

Fig. 6. The algorithm is stable even with complex images. This image is scaled to 1:1 and ready to be printed.

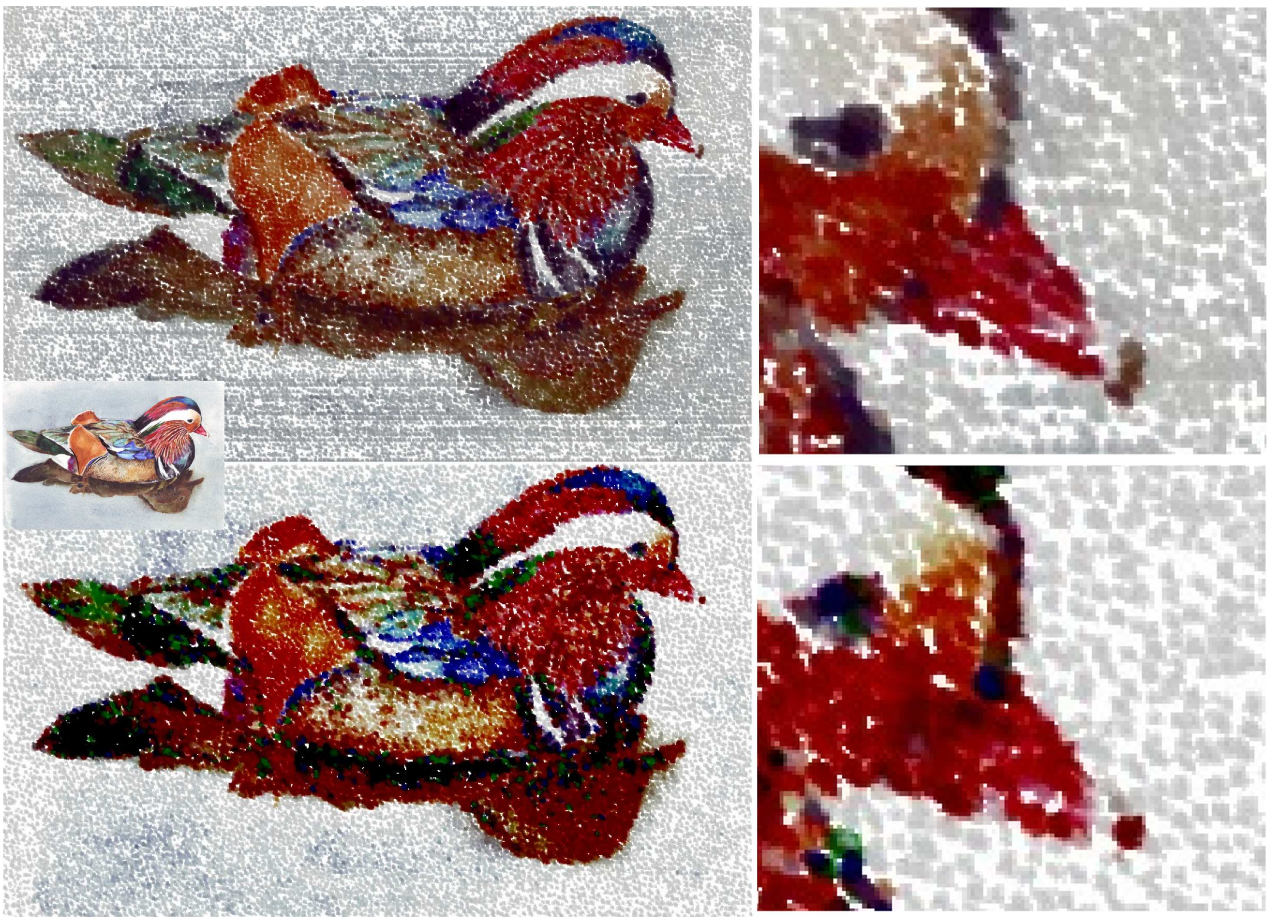

Fig. 7. The coeficients α, β and γ has been adjusted by hand on the top image; the image on the bottom has been generated enabling the expert system. An augmented part of the illustrations is shown on the right.

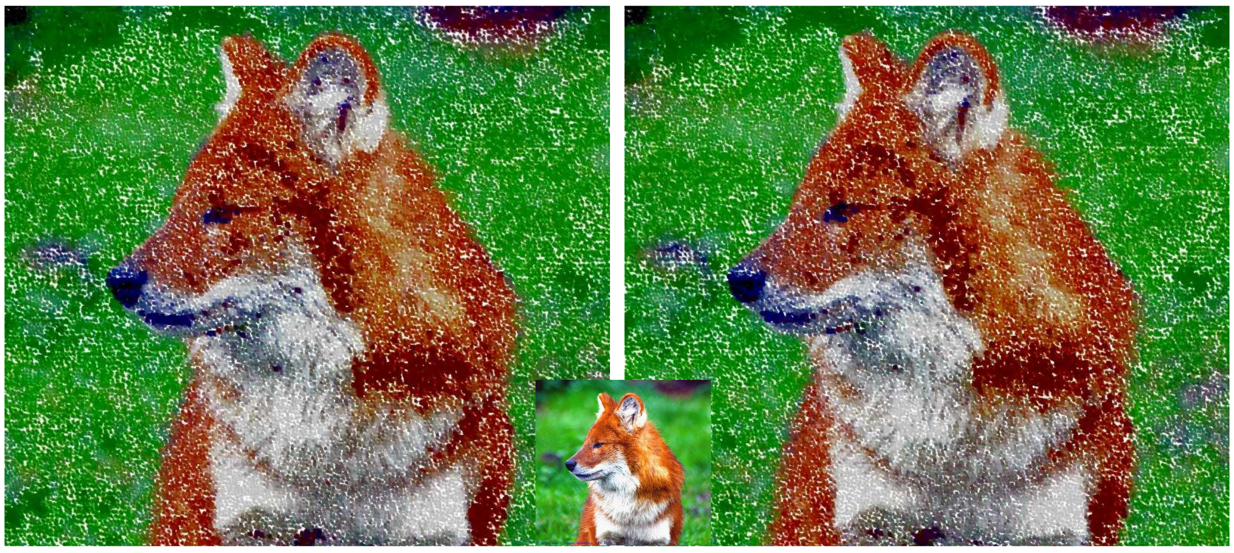

Fig. 8. These images are the result of different executions of the algorithm. As we can see the result is almost identical.