AUTOMATIC COMPUTER GENERATION OF STIPPLED ILLUSTRATIONS WITH COLOUR FELT-TIP PENS

First Author Name, Second Author Name
Anonymous location
author@server, author@server

Keywords: Rendering, Non-Photorealistic Rendering, Painting-like rendering, Drawing.

Abstract: Nowadays, non-photorealistic rendering is an area that not only focuses on simulating what artists do or the tools they use, but also on generating new expressive tools for digital art. In this paper we present a new algorithm to generate beautiful stippling illustrations with colour felt-tipped pen from a photograph or an image. This technique is not currently used by artists due to technical limitations, and thus the algorithm may prove a useful addition to the tools used by an artist. We introduce a novel stochastic approach to place coloured dots in a specific order based on the information about the contrast, borders and the histogram of the input image. The system is able to generate an unlimited number of non regular synthetic coloured dots without the need for scanning. These dots will be composed in a specific order to generate the final illustration.

1 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. The technique should 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 allows overlapping of dots to shade the illustration, which pointillism does not permit.

Several problems arise when an artist attempts to stipple an illustration using colour felt-tip pens. The first problem is the limited choice of colours available. A second problem is that the amount of ink in the sac and the porosity of the tip makes stippling complete illustrations expensive, especially with medium-tip markers. The quality of the paper must also be considered, as thin paper cannot tolerate a great amount of ink and may therefore tear, whereas thick paper may cause the ink to spread so much that shapes become blurred and poorly defined.

These kinds of illustrations are visually interesting but they are very hard to produce. If artists were able to efficiently use this technique, they could produce highly aesthetic images that maintain detail and shading even with large dots. We aim to provide the tools to produce these kinds of illustrations using a computer and an input photograph or image.

There are a number of previous works about stippling(Maciejewski et al., 2007; Mould, 2007) 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 about the contrast, the borders and the histogram to define a set of cells with a certain probability. The algorithm uses this probability to render the dots in a specific order, adding detail in every iteration. The algorithm can be stopped at any moment by the user but it also detects when the illustration has finished, and thus stops automatically.

This paper is structured as follows: In Section 2 we discuss related works. In Section 3 we present an overview of our system. In Section 4 we explain the algorithms in detail. Section 5 discusses the results of our approach. The paper is concluded in Section 6.

2 PREVIOUS WORKS

Abstract representation of still images was introduced by Haeberli(Haeberly, 1990) using image colour gradient and user interactivity for painting. Hertzmann(Hertzmann, 1998) places curved brush strokes of multiple sizes on images for painterly rendering. The technique fills with colour by using large strokes in the middle of a region and progressively smaller strokes as one approaches the edges of the region. Shiraishi and Yamaguchi(Shiraishi and Yamaguchi, 2000) improve the performance of the above method approximating the continuous strokes by placing rectangular strokes discreetly along the edges to create a painterly appearance. Santella and DeCarlo(Santella and DeCarlo, 2002) use eye tracking data to obtain points on images and create painterly rendering with that information. There are good works for painting terrains(Coconu et al., 2006; Bhattacharjee and Narayanan, 2008), 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(Secord et al., 2002). Some methods propose using circles instead of realistic dots(Mould, 2007; Gooch and Gooch, 2001; Schlechtweg et al., 2005; Yuan et al., ; Lu et al., 2003). 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(Secord, 2002; Pastor et al., 2004). In these cases, the use of Central Voronoy diagrams produces easily recognisable patterns. Renderbots is based on the idea of particles but the results, when they simulate stippling, also have the same problems with patterns(Schlechtweg et al., 2005). No works use coloured dots because it is not a common technique.

In the next Section, we will describe our solution for stippling with simulated colour felt-tip pens.

3 OVERVIEW

First, it is necessary to discuss the proposed system, which is based on the scheme presented in Figure 1. The algorithm has the following steps:

```
1: The information is obtained from the input image.
2: A matrix of probability is generated from this information.
3: The algorithm enters in a loop:
4: loop
5:    The place and size of the dot is computed based on the matrix.
6:    A new dot is generated with a simplified colour of the region.
7:    The dot is composed in the output image.
8: end loop
```

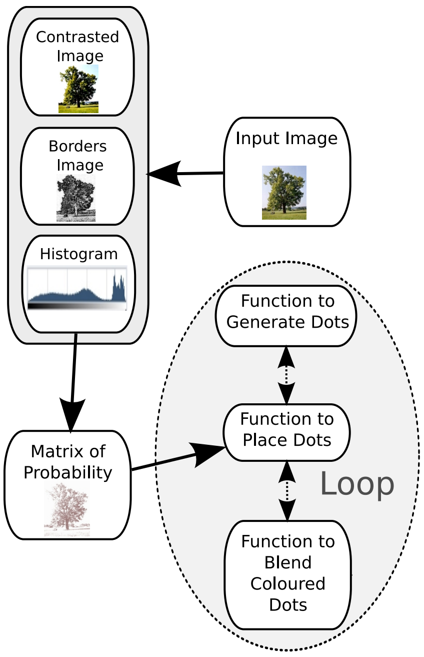

Figure 1: An overview of the algorithm.

The matrix of probability guides the dot placing process in such a way that it determines the order of dot placement.

Once the matrix has been generated, the algorithm for dot placement searches for a local maximum in the matrix by using a stochastical algorithm. If the local maximum is not 0, a dot is placed there. If the local maximum is 0, the algorithm repeats the search again until a specified number of iterations is reached, and then it stops. The user also has the ability to stop the algorithm at any time interactively.

Once the dot has been placed, the matrix is updated to remove a certain amount of probability, hence the probability of a cell decreases each time a dot is placed in the position given by the matrix of probabilities.

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

4 ALGORITHMS

The algorithm can be divided into several subalgorithms that take an input and return an output to the next step. These subalgorithms are explained in the following subsections.

4.1 Obtaining information

The first step is to filter the photograph in order to obtain relevant information before the algorithm is used to place dots. This 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 by marking silhouettes and stippling them. The next step is to stipple the darkest areas of the image. Finally, it is 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 tones in the photograph are ignored by the artist. Colour stippling is also guided by these rules.

Therefore, the extracted information is as follows:

- I_b: the image with the magnitude of the borders of the image.
- I_c: the contrasted image.
- H: the histogram of the image.
- m_H: the maximal value of the histogram.

I_b, I_c, and the input image are scaled to a fixed resolution for printing at the desired size. Dots are also generated according to the resolution. Once this step has finished, we have all the information needed to generate the matrix of probabilities.

4.2 The Generation of the Matrix

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

To generate the matrix (M), the following algorithm is computed:

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

Where α, β and γ must be adjusted according to the desired order for stippling the illustration. The values can be easily normalized if $\alpha + \beta + \gamma = 1.0$. These values are adjusted to simulate the process that artists follow, which has already been explained in Subsection 4.2: $\alpha = 0.4$, $\beta = 0.5$ and $\gamma = 0.1$. $Width$ and $Height$ are the size of the matrix M.

If the value of the histogram is greater or equal to m_H, then the value of the matrix becomes 0: $M[x,y] \leftarrow 0$.

All the values of the matrix can be normalized. Other thresholds can be included if more information about the image is obtained, for example, a mask can be taken as input and the probability of undesired areas of the image removed. The size of the dot can be stored in the matrix in a similar way: $M[x,y] \xleftarrow{size} 0.5 * I_c[x,y] + 0.5 * I_b[x,y]$

Once the matrix has been generated, the next step is to iterate until all the dots are placed.

4.3 Algorithm for Positioning Dots

The algorithm for dot placement removes the patterns of the output image. We use a method that is based on a Rejection Sampling Algorithm but includes some parameters that are controlled by the user.

The algorithm we propose contains the following steps for placing each dot:

```
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 output image
20: Update the matrix $M$
```

The algorithm takes the following parameters as input: N_ITERS and N_TESTS. N_ITERS indicates the maximal number of iterations of the algorithm before deciding that there are no more places to stipple. N_TEST indicates the maximal number of times to search for a higher probability. The inner loop searches for a local maximum whereas the outer loop searches for the probability in other areas. max_p indicates the maximal probability found, whereas $iteration$ indicates the number of iterations before the algorithm decides that a local maximum is found.

It is clear that the algorithm would never finish if the values of M did not change, and for this reason, the matrix must be updated. When a dot is placed on the output image, a value is subtracted from the image. This value depends on the intensity of the dots placed, therefore the central cells of the dot subtracts a higher value than the outer cells. Additionally, a certain amount is subtracted from the neighbouring cells of the pixel, as shown in Figure 2. This value can be adjusted in function of the distance from the middle of the dot. The radius of the affected neighbourhood is determined by the value of the contrast in the cell.

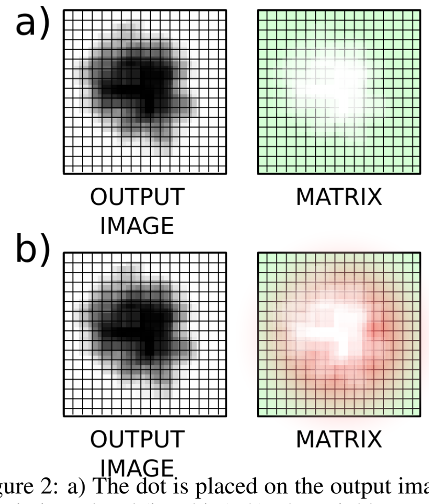

Figure 2: a) The dot is placed on the output image, and the matrix is updated (in white); b) The neighbour cells are also updated (in red).

The dots can be placed on the output image using the following equation for every pixel of the dot:

$$C_d = (1 - A) \cdot C_d + (A) \cdot C_s \tag{1}$$

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

4.4 Algorithm for Generating Colour Dots

The automatic generation of dots removes the artificial appearance of an illustration. Before generating a dot, the algorithm must decide the colour for the entire dot. The fact that each individual dot cannot contain more than one tone must be taken into consideration.

The colour is determined by a simple equation:

$$C_{dot} = \frac{C_{pixel} \cdot 3.0}{(C_{pixel}.red + C_{pixel}.green + C_{pixel}.blue)} \tag{2}$$

Where C_{dot} is the colour of the dot, and C_{pixel} is the colour of the related pixel in the source image. This normalization of the colour produces a flattening and a reduction of the brightness, which is used in algorithms of computer vision to simplify the illumination of the scene(Finlayson et al., 1998). Additionally, if the colour is almost black, the luminosity can be increased, especially if the intention is to print the image. As grey colours do not contribute much detail and can be easily produced by stippling repeatedly in the same place, grey colours are substituted by just two tones of grey.

The problem is reduced to generate grey scaled dots, and then multiply the obtained colour by the levels of intensity of the dot. This step is repeated for each dot placed on the output image. White values are not a problem because white pixels are not stippled by the algorithm.

The proposed algorithm to obtain grey dots is based on Monte Carlo methods(Jensen and Christensen, 1995). In these methods the algorithms decide when the solution is false, but do not know when the solution is true. Therefore, the algorithm iterates a certain number of steps until it reaches an approximate solution.

Our algorithm is as follows:

```
1: Create a matrix ($P$) of local probabilities
2: Create a matrix ($I$) of intensity values
3: Create a matrix ($A$) of absorption values
4: Generate a seed
5: **for** i = 0 **to** $i <$ N_DROPS **do**
6:     $F_b(n)$ returns a position $(p_x, p_y)$
7:     Deposit the ink in the returned position
8: **end for**
9: Apply a Gaussian filter to smooth the result
```

N_DROPS is the number of iterations of the algorithm, which can be estimated from the desired size of the dot. The size of the matrices P, I and A is twice the size of the dot. The matrix P is a probability density function (PDF). Therefore, P satisfies the discrete condition $\sum_{i,j} P(i,j) = 1$. I is the output image of intensities, whereas A is the absorption of the paper, and it can be obtained from a gradient image.

A small circle is a valid seed for the algorithm. This circle writes the matrix I with a dark value at its centre. The values of the matrix P are initialized to 0. However, seed cells are initialised to an uniform value, which is scaled according to the number of cells that comprise the seed.

A list (L_p) can be used to optimize the search for dots with probabilities different from 0. This list stores the positions of the cells with a probability distinct from 0. This list is updated in each iteration, removing the cells that becomes 0.

Once the seed and its neighbours have

been computed, ink is accumulated based on the function F_b in every step. The algorithm to compute the function $F_b(n)$ is the following:

```
1:  $i \leftarrow 0$
2:  $x \leftarrow Random(0,1)$
3:  while $x \neq 0$ do
4:      Get $p_x$ and $p_y$ from $L_p[i]$
5:      if $x \leq P[p_x,p_y]$ then
6:          RETURN $(p_x,p_y)$
7:      else
8:          $x \leftarrow x - P[p_x,p_y]$
9:          $i \leftarrow i+1$
10:     end if
11: end while
```

The function returns the position where the ink is deposited. The matrix I is used to accumulate the ink. The values of the ink and the values of I are normalized.

The viscosity of the ink and the absorption of the paper can be easily taken into account if we add these lines to the previous algorithm:

```
1: if $I[p_x,p_y] \leq A[p_x,p_y]$ then
2:     $c \leftarrow P[p_x,p_y]$
3: else
4:     $c \leftarrow P[p_x,p_y] \cdot v$
5: end if
6: if $x < v$ then
7:     $P[p_x,p_y] \leftarrow P[p_x,p_y] - c$
8: end if
```

v is a constant of viscosity. When a cell and its neighbours have a probability equal to 0 while the grey value is not 1, the criteria of expansion is applied recursively until any cell admits the shared probability. If no cells with probability distinct from 0 are found, it means that the paper cannot admit more ink, and the algorithm then finishes.

A resume of the algorithm is shown in Figure 3: a) The seed is placed on the image matrix (I), the probability is updated in the matrix P, all the cells of the seed have the same probability (in white); b) In the second step, the algorithm has filled a cell or the viscosity modifies the probability in P, the next drop of ink falls on a neighbour cell; c) After several steps, the algorithm forms the dot with a gradient from black to white.

The output dot is the generated dot of the algorithm described in the previous Subsection.

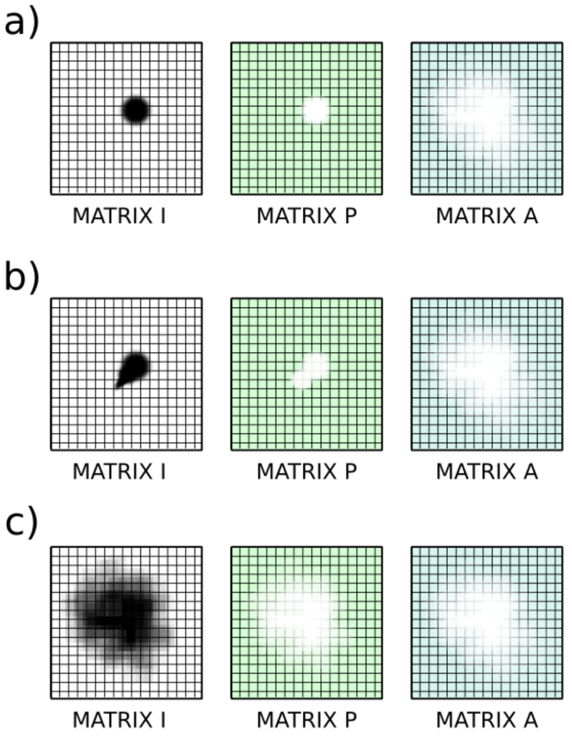

Figure 3: a) First step of the algorithm; b) Second step of the algorithm; c) After several steps of the algorithm, the dot is formed.

5 RESULTS

The algorithm produces a stippling illustration with colour felt tip pens that aesthetically grades the colours, as shown Figure 4. The algorithm is able to detect how many dots are placed on the output illustration by using information of the contrasted image. If the image background has an overall uniform tone, it is identified and removed, as can be seen in Figure 5. The algorithm can also take a rendered image from a 3D scene or a picture as input, which produces beautiful results that are shown in Figure 6. The algorithm also works with complex photographs, as shown in Figure 7 that has been correctly scaled for printing on a colour printer. The algorithm produces good results even with very complex photographs due to the distribution and size of the dots.

All these results have been generated with felt-tip pens between 40mm and 60mm.

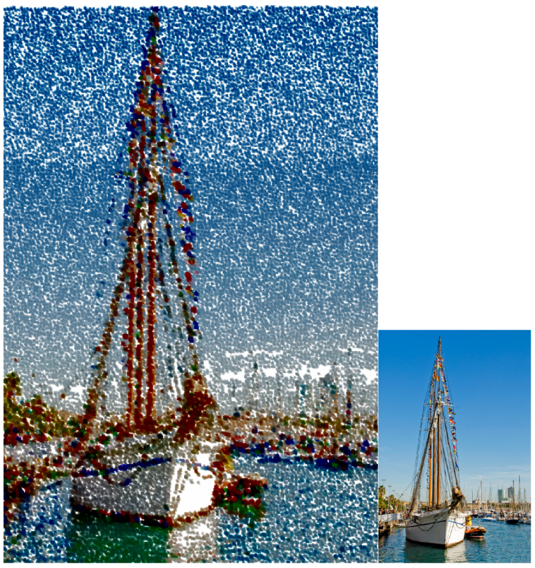
Figure 4: A result of our algorithm from photograph.

6 CONCLUSIONS AND FUTURE WORKS

We have presented an algorithm that automatically draws colour illustrations and simulates stippling with felt-tip pens. We have developed a fast and direct equilibration technique that is based on a probabilistic model. The algorithm places the dots in a natural way and in a specific order, just artists do. We have introduced an automatic control that allows the algorithm to detect when the illustration has finished. Our system provides both interactivity and high-quality output. It can utilise photographs, 3D rendered images and illustrations as input.

Future research into how more artistic knowledge can be included automatically within the application is proposed. Another important issue of future work is the introduction of temporal coherence when applying the algorithm to the frames of a video, especially when the intention is to maintain the same colours while objects are moving in a scene.

REFERENCES

Bhattacharjee, S. and Narayanan, P. J. (2008). Real-time painterly rendering of terrains. In *Proceedings of Sixth Indian Conference on Computer Vision, Graphics & Image Processing*.

Coconu, L., Deussen, O., and Hege, H. C. (2006). Real-time pen-and-ink illustration of landscapes. In *NPAR 06: Proceedings of the 2nd symposium on Non-photorealistic animation and rendering*, pages 27–36.

Finlayson, G. D., Schiele, B., and Crowley, J. L. (1998). Comprehensive colour image normalization. In *ECCV '98 : European conference on computer vision*, volume 1407, pages 475–490.

Gooch, B. and Gooch, A. (2001). *Non-photorealistic Rendering*. A. K. Peters.

Haeberly, P. (1990). Paint by numbers: abstract image representations. In *SIGGRAPH'90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 207–214.

Hertzmann, A. (1998). Painterly rendering with curved brush strokes of multiple sizes. In *Computer Graphics (Annual Conferences)*, number 32, pages 453–460.

Jensen, H. W. and Christensen, N. J. (1995). Photon maps in bidirectional monte carlo ray tracing of complex objects. *Computers and Graphics*, 19(2):215–224.

Lu, A., Morris, C., Taylor, J., Ebert, D., Rheingans, P., Hansen, C., and Hartner, M. (2003). Illustrative interactive stipple rendering. In *IEEE Transactions on Visualization and Computer Graphics*, volume 9, pages 127–138.

Maciejewski, R., Isenberg, T., Andrews, W. M., Ebert, D. S., and Sousa, M. C. (2007). Aesthetics of hand-drawn vs. computer-generated stippling. In *Proceedings of Computational Aesthetics in Graphics*, number 3, page unknown.

Mould, D. (2007). Stipple placement using distance in a weighted graph. In *Proceedings of Computational Aesthetics in Graphics*, number 3, page unknown.

Pastor, O. M., Freudenberg, B., and Strohthotte, T. (2004). Real-time animated stippling. In *Proceedings of NPAR 2004*, volume 23, pages 62–68.

Santella, A. and DeCarlo, D. (2002). Abstracted painterly renderings using eye-tracking data. In *NPAR 02: Proceedings of the 2nd symposium on Non-photorealistic animation and rendering*, pages 53–58.

Schlechtweg, S., Germer, T., and Strothotte, T. (2005). Renderbots: Multi agent systems for direct image generation. *Computer Graphics Forum*, 24:283–290.

Secord, A. (2002). Weighted voronoi stippling. In *Proceedings of NPAR*, pages 37–43. ACM Press.

Secord, A., Heidrich, W., and Streit, L. (2002). Fast primitive distribution for illustration. In *Thirteenth Eurographics Workshop on Rendering*, pages 215–226.

Shiraishi, M. and Yamaguchi, Y. (2000). 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*, pages 53–58.

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

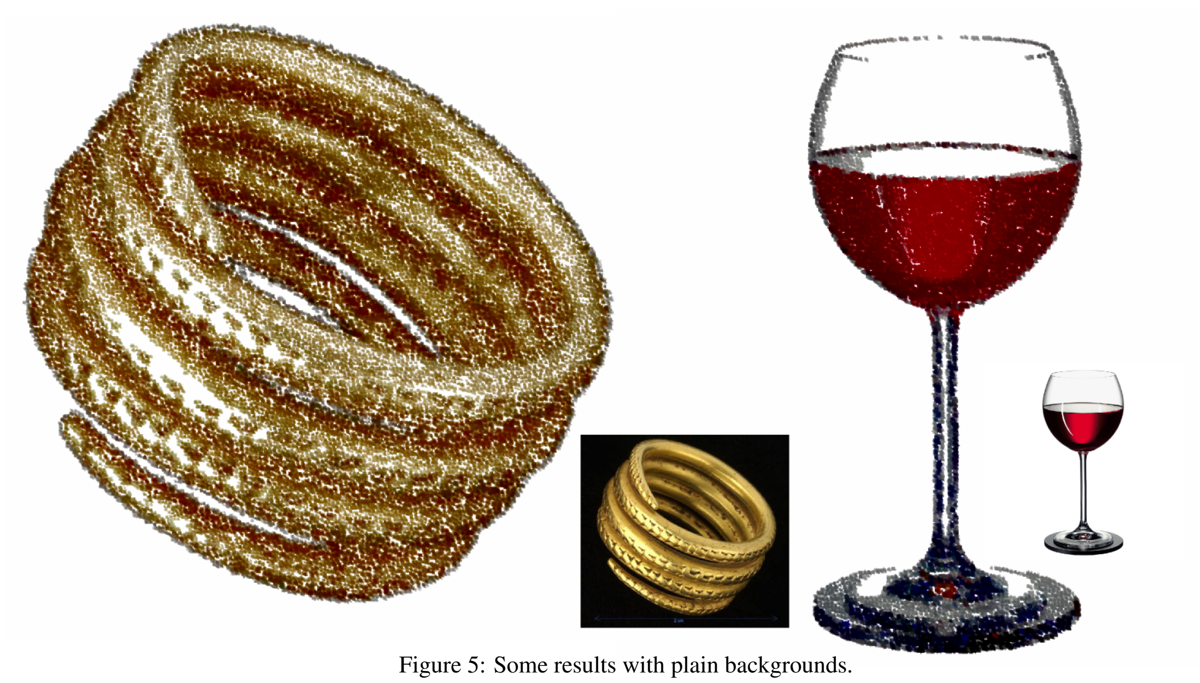

Figure 5: Some results with plain backgrounds.

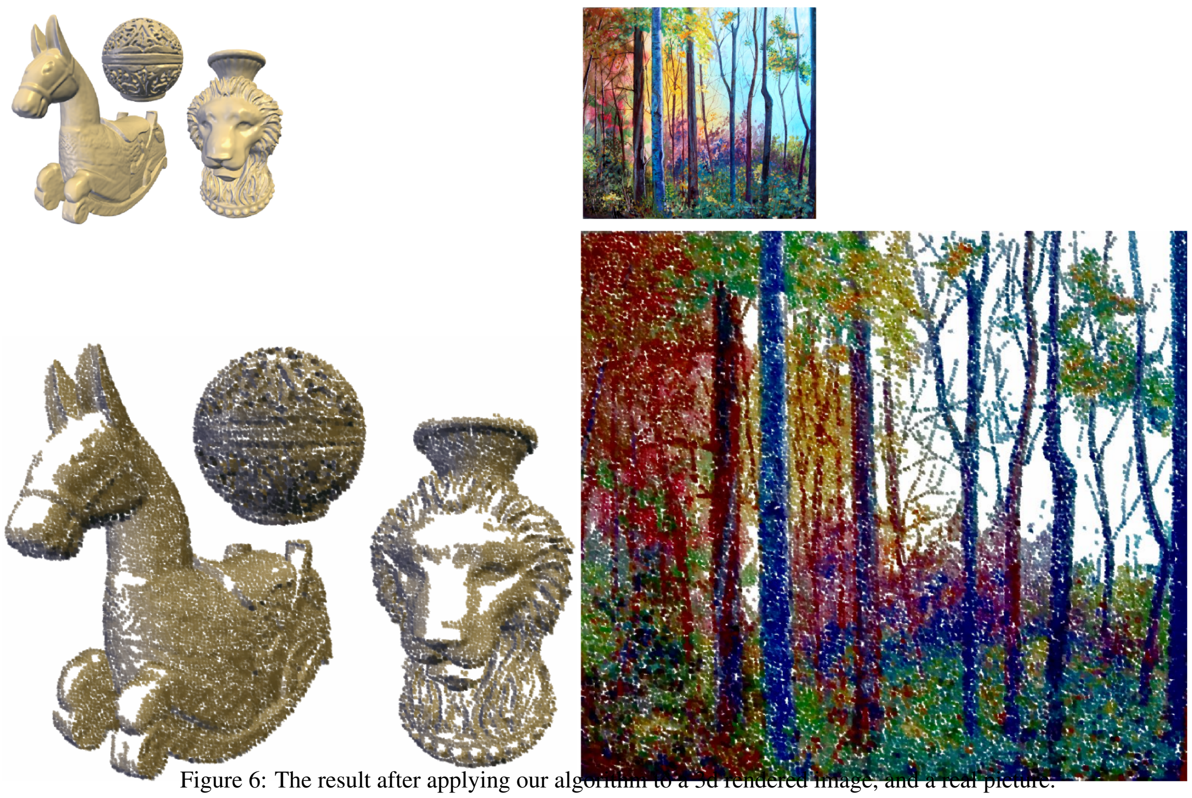

Figure 6: The result after applying our algorithm to a 3d rendered image, and a real picture.

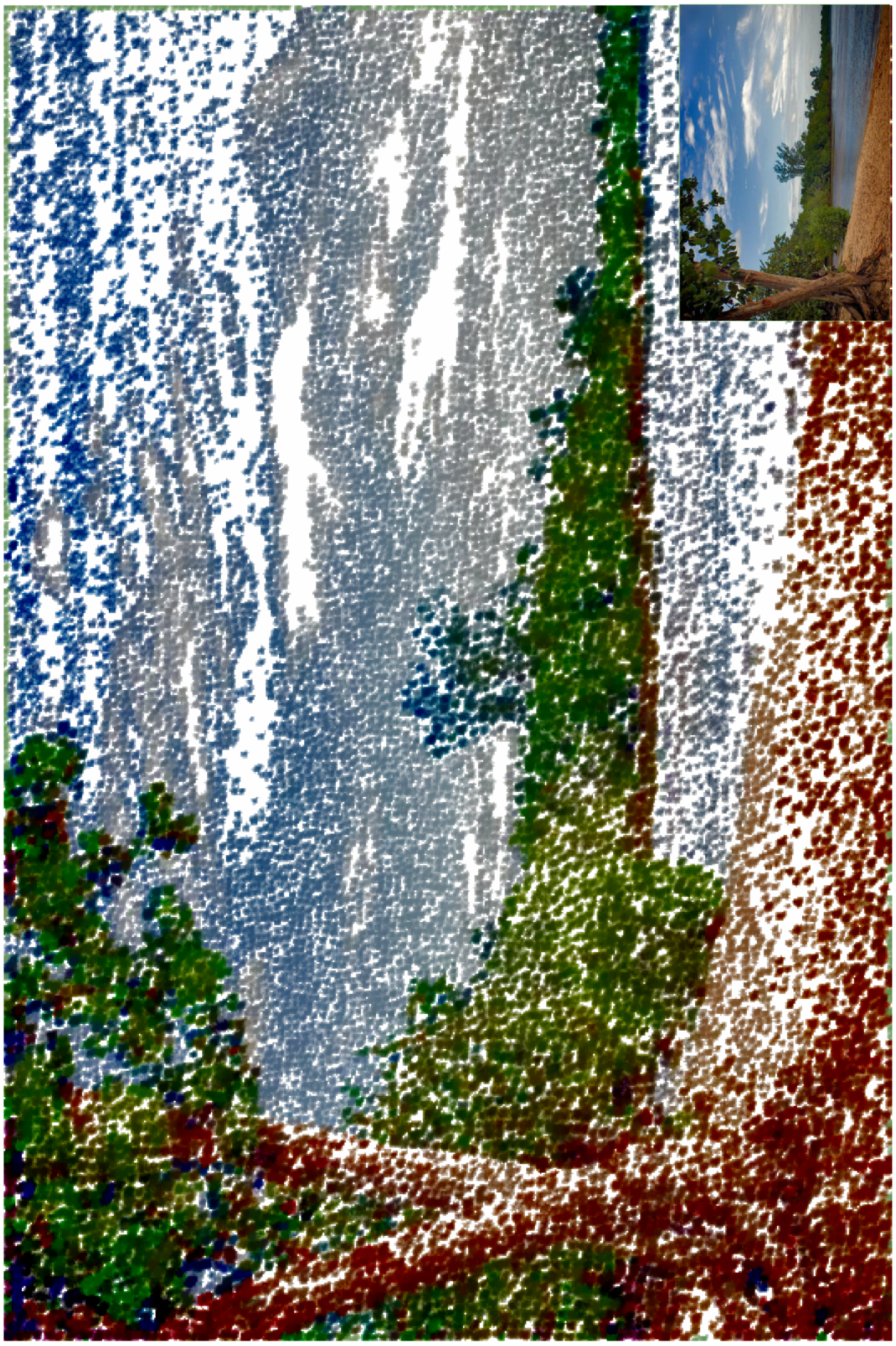

Figure 7: An illustration with a scale of 1:1, ready to be printed.