
Special Issue: CAD in the Arts

Guest Editor: Carlo H. Séquin, University of California, Berkeley

# Stochastic Generation of Dots for Computer Aided Stippling

Germán Arroyo[1], Domingo Martín[2] and M. Victoria Luzón[3]

[1]University of Granada, arroyo@ugr.es
[2]University of Granada, dmartin@ugr.es
[3]University of Granada, luzon@ugr.es

**ABSTRACT**

Current computer stippling methods experience problems with the placement and the appearance of dots. In this paper we present a solution to both problems with a new method to generate synthetic yet realistic dots, and a new algorithm for dot placement, solving problems of overlapping and the formation of patterns, and allowing a full tonal range. Both solutions are based on probabilistic algorithms. The method runs in real time allowing the illustration to be modified while it is being created. This is possible by means of a simple set of tools, which are mapped to high level operators allowing the application of artistic techniques to improve the final result.

**Keywords:** non-photorealistic rendering, stippling, hand-drawn half-toning.
**DOI:** 10.3722/cadaps.2010.447-463

## 1 INTRODUCTION

Stippling is a pen-and-ink technique using small dots to represent shape, tone and texture. Stippling should not be confused with half-toning. Images produced by half-toning or dithering, and computer printers, operate on similar principles (varying the size and/or spacing of dots on paper) [1], but this process is mechanical, whereas stippling simplifies the visual information and has an hand-drawn appearance. Stippling has many advantages, for example, many natural science and archaeology journals use scientific illustrations due to the educational and communicative ability of this traditional style of illustration [2].

Previous methods have serious problems with the shape and appearance of dots, as they all make use of black dots or simple geometric shapes for the dots, unlike hand-drawn stippling that produces irregular gray-scale dots. They also experience problems when producing tonal ranges, due to the fact that all dots used are solid black. In relation to the first problem, most techniques do not simulate the dots correctly, or they scan dots from original illustrations. This is a tedious process that produces repetitive dots. With regards to the second problem, previous methods have a problem with overlapping dots on the paper when simulating gradients of tone. This occurs due to the fact that

these methods use completely black dots, producing strange overlapping or patterns. In addition, previous methods do not allow interaction with the user because they use computationally expensive algorithms that require a long time to produce the final illustrations.

In this paper we propose a solution to these problems. We present an approach based on Monte Carlo algorithms to generate realistic looking dots for stippling. We also propose a new method that places dots in a way that prevents patterns and produces a realistic range of tones. The method also allows user interaction by varying the probabilities of placing dots and the way they are generated. The method works in real-time, providing a user interface that can modify the appearance of the illustration. The tools presented in this paper simulate the action of artists when they stipple illustrations.

## 2 RELATED WORK

There has been much research concerning stippling. We must first distinguish between those methods that use 3D polygonal models as input of the algorithm to produce stippling and those that use 2D images as input. This is important because in the case of 3D models, the user or artist has complete control over the conditions of the scene: camera, lighting, colors, etc. By contrast, the user or artist is much more limited with 2D images, the only possibility being to change the color of pixels. The best source is therefore a 3D model, but then it is more involved to create a picture.

Starting with works that use a 3D model, Pastor et al [3] used a method similar to edge collapse [4] for locating dots; allowing temporal coherence, this method works with different frames of a video as input. The method places the dots without randomness; this allows the dots stay in the same position when different frames are involved. Lu et al. [5][6] developed a method to achieve volume rendering by using dots based on the gradient computation between voxels. The method proposed by Sousa et al. [7] obtains information from the original triangle mesh, which is used to draw different primitives, including dots. Xu and Chen [8] used scanned data to produce stippling. Zakaria and Seidel [9] used the silhouettes as a visual complement, and geometric primitives for dots. Yuan et al [10] used an intermediate space between the image and the 3D model, called geometry-image, to obtain different properties of the model.

We are especially interested in methods that produce stippled renderings from 2D images. Among these methods, Secord [11], Deussen et al [12], Hiller et al [13], Dalal et al [14] and Barla et al [15] used the Centroidal Voronoi Diagrams (CVD) to distribute the dots. Their approach has problems: CVD distributes the dots without the possibility of overlapping, and dots are drawn as circles or other simple geometrical shapes. Secord et al [16] also used a variant of the previous method to draw dots and other primitives using probabilistic density functions; the main problem with this technique is that the density functions cannot vary while the algorithm is executing. It is difficult to develop a user interface that interacts with these algorithms, while at the same time allowing real-time interaction. Schlechtweg et al [17] developed a multi-agent system called Renderbots, to position different kinds of drawing primitives. Although the system seems very efficient with general drawing, patterns can be readily observed. Kopf et al [18] used Wang Tiles to create and locate dots with a blue noise function in a set of tiles, but the problem is that the method produces errors in the borders of the tiles, which must be fixed. Also, the dots they used are completely black. The algorithms presented in Mould [19] are based on a weighted graph that controls the location of dots, enhancing the borders. Although it produces a blue noise distribution, the results show an alignment of dots close to the edges. Kim et al [20] used a feature flow to align the dots following the shapes. This kind of stippling is not used in technical illustration, due to the patterns it produces. Kim et al [21] provided a method for extracting a tone map and dots from the original image. It applies a statistical analysis and produces a synthetic stipple distribution, which is used to obtain the final result. The method provides a way of synthesizing human styles. While providing good results, it is complex and has some difficulties with the automatic extraction of dots, with the generation of the synthetic textures, and with the generation of a continuous tonal range, due to the fact that only black dots are used.

The work of Isenberg et al [22] should be highlighted. This observational study focused on the differences between hand-made and computer-made stippled drawings. An important result was that both types of drawings could easily be distinguished as either hand-drawn or computer generated. In

the case of computer generated images, they were clearer and more similar than hand-drawn images. Another important conclusion was that computer generated stippling uses very regular dots. The artifacts and patterns produced when placing dots with traditional computer programs are also an undesirable result. Maciejewski et al [23] used a statistical texture analysis to compare the underlying mathematical differences in the structure between hand-made and computer-made images. This work is complemented and extended in [24].

## 3 STUDY OF TRADITIONAL STIPPLING ILLUSTRATIONS

In this section components involved in stippling illustrations are explained. We have classified these components into two groups: a study of the techniques used by artists and the study of the dots they usually produce. Both of them are presented in detail in next subsections.

### 3.1 Techniques used by Artists

We asked several artists to draw some stippled illustrations and asked them about the way they work, the techniques they use and the results produced in their final illustrations. Based on their descriptions we have classified these techniques according to the way the artists work into a set of high level tasks, which can be implemented as operators for algorithms to place dots. The different techniques have been marked with different numbers in Figure 1:

1. *Elimination of parts:* One of the most frequently used techniques is the elimination of parts or details in the image. This consists of removing some unimportant parts of the image in order to focus the observer's attention on other areas. In this example, the artist has removed all the sky and part of the scenery.
2. *Simplifying complex parts:* At times, it is impossible to remove some parts because they are not important but they cannot be completely removed. When this is the case, a reduction in rendering detail is necessary. In the example, it is done by stippling the borders only of the less important areas.
3. *Increasing contrast:* The example shows how the artist has changed the level of detail by changing the contrast of the image. The artist uses a global evaluation of luminosity and a local evaluation of contrast to enhance the details in very bright and very dark areas. Where visual information does not exist, the artist invents it using her knowledge of the 3D model or scene.
4. *Color inversion:* As can be seen in Figure 1, the color inversion is a very useful technique that allows the detail of some parts of the image to be increased. The artist changes the darker parts by inverting contrast with appropriately chosen stippling. This is not a general rule, so the artist only uses this technique to enhance very specific parts of the image.
5. *Smoothing shapes:* Another important technique is altering regular forms by smoothing them.
6. *Enhancing the detail:* It is possible to see here how the artist has drawn silhouettes and redefined the shape of the stones that comprise the monument. Again, the artist has used personal knowledge to solve the problem.

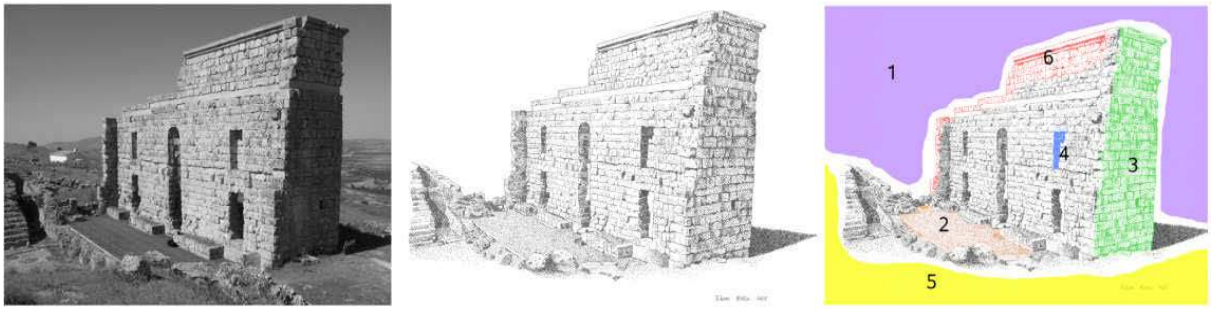


Fig. 1: From left to right: a photograph; the illustration from the photograph; the marked areas with different techniques applied by the artist.

### 3.2 Study of the Dots

When an artist places the pen on the paper, the position of the dot is not exact. It depends on the speed, the size of the pen, the strength, the style and the level of detail.

We scanned dots with a resolution of 600dpi in gray-scale, as this resolution allows correct visualization of various dithering algorithms on a screen and in printer output. We have observed that the dots are not completely black; they contain many gray values. The darkness depends on the time the pen is in contact with the paper and the properties of the ink. This is significant and gives each dot a different size, shape and darkness. The deposited ink is absorbed by the paper, toning down the color of the ink.

With regard to the distribution of the analyzed intensities of the pixels and the shape of the scanned dots, we have classified them according to these characteristica:

- *Regular dots:* dots that have a single dark kernel approximately in their center.
- *Irregular dots (or non uniform dots):* the dark kernel is not in their center.

Regarding to the shape we have used this classification:

- *Mono-kernel dots:* have only one kernel.
- *Multi-kernel dots:* have more than one kernel.

Some dots can be seen as mono- or multi-kernel, and as regular or as irregular at the same time. The first dot in Figure 2 is both regular and mono-kernel.

Fig. 2: From left to right, different scanned dots (600 dpi): A regular dot; an irregular dot; a mono-kernel dot; a multi-kernel dot.

The physical parameters that affect the shape and intensity of the dots are the following:

- *Viscosity of the ink:* The viscosity is the resistance produced by the cohesion of the particles that compose the ink. These particles produce an internal friction that slows their movement. The equations of displacement can be simplified with an arbitrary, but constant resistance force for dots [25].
- *Absorption of the paper:* The paper is composed of interlaced fibers that absorb liquids. In the case of the ink, the pigments lay over the fibers, dyeing the paper [26]. Areas with higher absorption contain more pigment, and thus appear darker.
- *The size of the dot:* The size of the dot depends on the above characteristics and also on the size of the pen, which is usually between 0.2 and 0.4 millimeters. Artists may use dots of different sizes depending on the effect that they want to produce. However, they normally use the same size pen for one illustration. The size of the dot varies from dot to dot and it depends not only on the size of the pen, but also on the time the artist presses the pen on the paper. The regularity of the dots also varies depending on the movement of the artist when drawing the illustration. This can be considered as a stress parameter for the dot.
- *Color of the ink:* The ink used in stippling is usually black. When the technical pen touches the paper, the deposited ink spreads over the paper and the dot is toned down from black to the color of the paper. The color depends on the amount of pigments and/or dyes in the ink. When ink is not black, its color depends on the base tone, but the effect is the same.
- *Amount of pigments and/or dyes*: The amount of pigments and/or dyes affects the density and the color of the ink. There is a direct relationship between the quantity of pigment and ink color. Taking into consideration that the amount of ink deposited on the paper is finite, and that the probability that some amount of ink marks the paper depends on various physical parameters, a statistical analysis is required.

The amount of dye is directly related to the darker or lighter appearance of the areas of the dot. We can define a statistical function that describes this effect. This function is determined by a

Probabilistic Density Function (PDF). The PDF can be computed independently for each dot, and it is equal to the function defined by the intensity of the scanned pixels for each dot. In this case, the probability of placing ink with the technical pencil is a 2-dimensional function. The PDF is an asymmetric function for irregular dots, and is a multi-modal function for multi-kernel dots. Multi-kernel dots may be obtained by superposing several uniform dots, and irregular dots can be obtained by altering the regularity of the PDF. In Figure 2 these characteristics are shown in accordance with the previous classification.

Let $f(x)$ be the PDF in two dimensions, therefore $\int_{-\infty}^{\infty} f(x,y)dxdy = 1$ is satisfied.

This PDF guarantees that there is no loss of probability along the function due to the normalization produced when the probability changes in some part of the function; this means that the sum of all probabilities is always 1. In any case, a higher probability of deposing any ink on the paper occurs on the darker parts of the dot, whereas the lighter parts of the dot express a lower probability of having ink deposed.

The probability of deposed ink is similar to a normal distribution for regular dots, but not in others. This variety seems to indicate that there are no simple equations to generate all kinds of dots; therefore a statistical approach is required. We have classified the dots according their intensity and shape in order to compute an algorithm that generates all kinds of dots automatically.

## 4 COMPUTED-AIDED STIPPLING

The proposed solution for the simulation of realistic dots, the placement of the dots, and the user interface of the system are presented in this section. Due to the fact that in manually stippled illustrations dots do not all have the same appearance, and two illustrations of the same photograph can be different while maintaining the same global shape and shading appearance, we have decided to use a statistical approach. This approach allows the generation of an unlimited set of similar dots and a random (but controlled) placement for the dots. Our approach also solves the problems of overlapping and pattern generation by using dots with different levels of gray.

An overview of the system is shown in Figure 3. This figure shows that the first step is to generate a new statistical function $(F_{pdf})$ for dot placement according to the information of the image's borders, the contrast-adjusted image, and its histogram. These data are obtained from the input image. The parameters or the statistical function are preset at the beginning of the algorithm. The algorithm for dot placement uses this statistical function to place every dot. The shapes of the dots are generated by another statistical algorithm. When a dot is placed, $F_{pdf}$ also changes, which is possible because the function is mapped to a mask. This mask stores both the probability and the size of the dots for the illustration. The system iterates until the user stops the algorithm. The partial results are stored to compute the next iteration in $I_{out}$. When the user stops the algorithm, the partial result is the final result of the algorithm. When the user applies any operator provided through the user interface in any step of the algorithm, the partial result ($I_{out}$) as well as the mask are altered, which then changes the way the dots are placed.

The core of our method is the algorithm for placing dots and the algorithm for dot generation. Therefore, a very important element is the statistical function that is used when a new dot is placed in the illustration and that is related to the user interface. These algorithms are explained in greater detail in the following subsections.

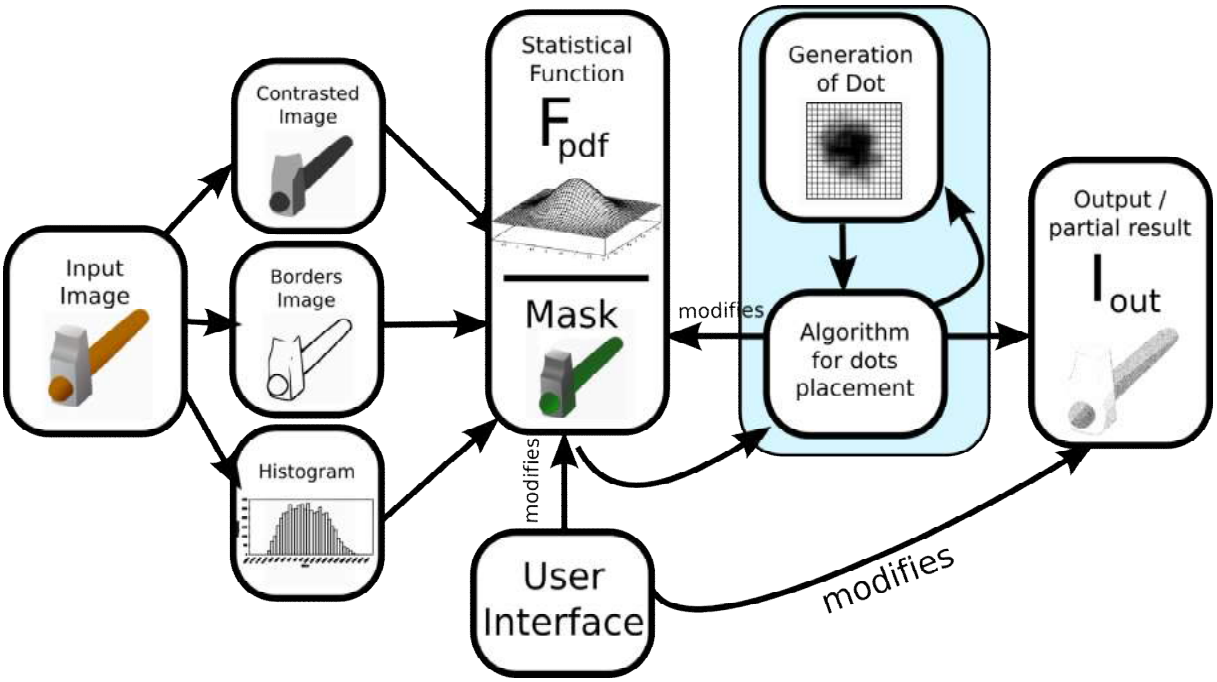


Fig. 3: Overview of the algorithms and their integration with the user interface.

### 4.1 Statistical Function ($F_{pdf}$)

The statistical function ($F_{pdf}$) gives the probability where to place dots in the image. $F_{pdf}$ is computed from the contrasted image, the image's borders and the histogram of the image using a simple function for every pixel of the image:

$$F_{pdf}(x,y) = a \cdot I_c(x,y) + b \cdot I_b(x,y) + c \cdot H(x,y)$$

Here $I_c$ is the contrast-adjusted image, $I_b$ represents the borders of the image, and $H(x,y)$ is the value of the histogram for the pixel in the position $(x,y)$. $I_c$ is obtained with a simple filter that modifies the gray values of all pixels to obtain a better looking shading (for example multiplying every pixel by itself); $I_b$ is easily obtained using a Sobel filter [27]. The values $a$, $b$, and $c$ are three constants that are preset at the start of the algorithm. Three values obtained experimentally that work well in most cases are: $a = 0.5$, $b = 0.4$, $c = 0.1$. This means that dots are guided by the contrasted image with a 50% probability, 40% by the image of borders, and 10% by the histogram. This allows enhancement of borders and areas with special details in the image without losing the general shape and shade of the input photograph.

Once a dot has been placed, the function decreases the probability in its neighborhood and spreads it to the rest of the neighbor values in $F_{pdf}$; therefore $F_{pdf}$ also determines the space between dots. The PDF ($F_{pdf}$) is a continuous function, discretized in a mask with the size of the output image ($I_{out}$). This mask guides the dots placement, and also stores the values of the size of the dots ($F_{size}$) for every value of $F_{pdf}$. These values are the same as the probability at the start of the algorithm: $F_{size}(x,y) = F_{pdf}(x,y)$. Therefore, the initial values of the mask are given by the evaluation of the functions $F_{size}$ and $F_{pdf}$ for every position of the mask given by $(x,y)$. This process is explained in detail in the next subsection.

### 4.2 Dot Placement

The algorithm for dot placement removes the patterns in the final image and solves the overlapping problem. The solution is to place the dots randomly according to the values returned by the statistical function, which contains information about the borders and shading of the input photograph. We use

a method that is based on a Rejection Sampling Algorithm, guided by the mask. This method basically searches for the best candidate from the previous mask, by randomly sampling a subset of the function and taking several local maxima. The algorithm then selects one of these local maxima, and places the dot at that position. If the selected maximum is less than or equal to zero, no dot is placed.

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

1. *iteration* <- 0; *maxP* <- 0
2. WHILE (*maxP* = 0 AND *iteration* < *N_ITERS*):
   - 2.1. FOR *i* = 0 TO *i* < *N_TESTS*:
     - 2.1.1. *x* <- Random(0, *Width* - 1); y <- Random(0, *Height* - 1)
     - 2.1.2. *p* <- $F_{pdf}(x,y)$
       - 2.1.2.1. IF (*p* is a new maximum) THEN:
         - 2.1.2.1.1. *maxP* <- *p*; s <- $F_{size}(x,y)$
   - 2.2. *iteration* <- *iteration* + 1
3. IF *maxP* = 0 THEN: End of the algorithm
4. Generate a dot of size *s*
5. Blend the dot with the final image
6. Update the values of the function $F_{pdf}$

The algorithm takes two parameters as input: *N_ITERS* and *N_TESTS*, and the size of the input image is given by *Width* and *Height*. *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. A pre-set given by *N_ITERS* = 1.000 and *N_TESTS* = 10.000 produces good results; these values have been empirically obtained. Greater values take longer to decide where to place a dot, whereas lower values tend to blur the final result. The inner loop of the algorithm searches for a local maximum, whereas the outer loop skips to another area to search for the highest probability. First a series of candidates is randomly generated after a local and a global search. From these candidates the highest value is taken. A dot is computed then and placed in the position given by the best candidate. Finally, at third step, the probability of the best candidate is decreased in the matrix of probabilities. *maxP* indicates the maximal probability found, whereas *iteration* indicates the number of iterations before the algorithm decides a local maximum has been found. It is clear that the algorithm would never finish if the values of the $F_{pdf}$ did not change. For this reason the matrix must be updated. When a dot is placed on $I_{out}$, a set of values is subtracted from the values of the mask. These values depend on the intensity of the placed dot. Therefore, dark pixels of the dot subtract a higher value than lighter pixels. Additionally, a certain value is subtracted from neighboring values of the pixel. This value is a radius that can be adjusted as a function of the desired distance of this dot from other dots. The radius of the affected neighborhood is determined by the contrast in the related pixel. This value can be adjusted as a function of the distance from the center of the dot. The subtraction prevents the overlapping of too many dots in the same area of the image. The radius value of this subtraction can be easily adjusted as a function of the size of the placed dot; a typical value is 1.5 times the size of the dot.

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

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

Here $C_d$ is the color destiny, $C_s$ is the color source, and $A$ is the value of the intensity of the dot. The result of this algorithm is shown in Figure 4.

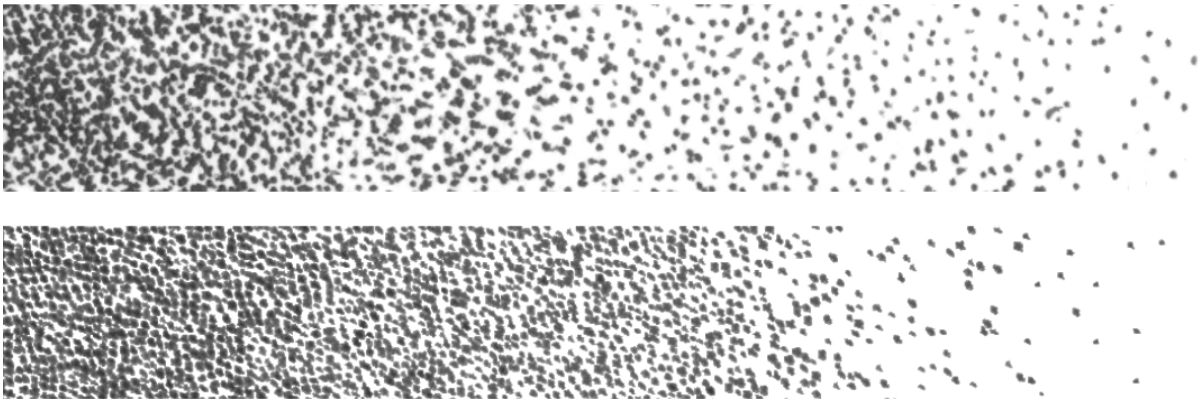

Fig. 4: A tone range drawn by the artist (top) compared with a similar tonal range generated by our algorithm (bottom).

### 4.3 Dot Generation

In this subsection we present our solution for generating an unlimited number of different dots for stippling based on a stochastic method.

The steps of the general algorithm are the following:

1. Generation of regular dots by using a Monte Carlo algorithm
2. Modification of the basic algorithm to superpose the kernels from several regular dots
3. Modification of regular dots to change the shape based on the speed of the artist when he or she places the dots

The proposed algorithm is based on Monte Carlo methods, which have been used extensively to generate realistic images. In these kinds of methods the algorithms decide at every step when a solution is false but cannot determine when a solution is true. Therefore, the algorithm iterates a certain number of steps until an approximate solution is reached. We know when the solution is false when trying to simulate the shape of a dot: it is false when there is no possibility of ink placement on the paper (for example, when the cells the of paper cannot admit more ink or when the ink cannot spread out due to its viscosity), but we do not know when the solution is true (where exactly the ink may have spread to). Therefore, a statistical function is required to place the ink drops on the paper. This function tries to obtain an estimation of the real solution (that is a real dot placed by an artist).

The Monte Carlo algorithm is an approach to an estimation of such a real dot. Our Monte Carlo based algorithm uses a matrix where the final result will be stored. This matrix is the matrix (I), and it stores the gray pixels of the dot. The algorithm has the following steps:

1. Create a uniform matrix (*I*);
2. Create a matrix (*P*) with the same size as *I;*
3. Select a seed;
4. While there is still ink:
   4.1. Generate a random position according to the function $F_b()$*;*
   4.2. Place the ink in the returned position and update the affected values in matrix *P;*
5. Smooth gray values: a Gaussian filter is applied to the final dots;

The matrix *I* stores values of gray tones of the accumulated ink; initial values in *I* are set to the maximal gray value. The values of I decreases in every step of the algorithm, controlling the darkness of its pixels and simulating the absorption of the ink by the paper. The matrix *P* stores values of probability; initial values in *P* are set to 0. Function $F_b()$ returns a position in the matrix, hence, $F_b()$ is the PDF. The created matrix has a size greater than or equal to 10*x*10 to allow the ink enough space to be spread. How to compute the function $F_b()$ will be explained after discussion of the generation of the initial seed.

### 4.3.1 Generation of the Seed

The seed is required to initialize the algorithm. It simulates the moment that the pen touches the surface of the paper for the first time. We can use a normal distribution *N()* to generate the seed. The normal distribution must be centered, that is to say that the mean of *N()* must be 0.5. Therefore, the seed of the dot is placed on the center of the matrix in most cases. The typical deviation is not particularly significant because it affects the dispersion of the seed along the matrix. The initial position of the seed is defined by using the function $(p_{xi}, p_{yi}) = N()$ in the grid.

The algorithm has an input parameter $n$, which is given by the user to create $n$ new positions around the initial seed. The value $n$ defines the size and the shape of the pen. A circle with same radius as the pen is a good approximation for seeds to generate regular dots. A line or several circles can be defined for multi-kernels dots. All the $n + 1$ positions (the initial seed is also included) have the same probability, which is computed using this simple equation: $P(p_{xi}, p_{yi}) = 1 / (n + 1)$. This probability is modified in the matrix $P$ for every iteration of the algorithm. The matrix $P$ returns the discrete evaluation of the PDF for every position. The matrix $P$ is a PDF because it satisfies the discrete condition:

$$\sum_{i,j} P(i,j) = 1$$

A quick way to search all the positions with the same probability is to use a list $Lp$ that stores the position of every value in the matrix $P$ that has a probability distinct from 0.

### 4.3.2 Computation of $F_b$

Once the seeds have been computed, ink is accumulated based on the function $F_b()$ for every step. The algorithm to compute the function $F_b()$ is the following:

1. $i$ <- 0
2. $x$ <- GetRandomBetween(0, 1)
3. WHILE $x \neq 0$:
   - 3.1. Get $p_x$ and $p_y$ from Lp [i]
   - 3.2. IF $x \leq P(p_x, p_y)$ THEN:
     - 3.2.1. RETURN$(p_x, p_y)$
     - 3.2.2. END OF THE ALGORITHM.
   - 3.3. ELSE:
     - 3.3.1. $x$ <- $x - P(p_x, p_y)$
     - 3.3.2. $i$ <- $i + 1$

Finally the function returns the position where the ink must be accumulated. The matrix $I$ is used to accumulate the ink. The ink has a color $k \in [0,1]$, and the values of the matrix $I$ also contain real numbers between 0 (black) and 1 (white). The ink is accumulated in the matrix $I$ by subtracting the value $k$. This simulates the still pen on the paper with an absorption of 100%. Obviously this kind of paper does not exist and the result is not realistic, so the algorithm needs some modifications that will be explained in the following subsections.

The number of iterations depends on the amount of ink we want to place, that is to say, the amount of time the artist leaves the pen on the paper.

### 4.3.3 Simulation of Paper and Viscosity

One problem with the previous solution is that paper that absorbs all the ink does not exist. Therefore, the absorption ($A$) is a relevant factor when generating dots. Moreover, the viscosity ($v$) of the ink is usually low, allowing the ink to spread over the paper with low absorption.

The previous algorithm is extended just before step 3.2.1. to take into account the viscosity and to change $P$ for each iteration:

1. $x_v$ <- GetRandomBetween(0, 1)
2. $c$ <- $P(p_x, p_y)$ □ $v$

3. IF $x_v < v$ THEN: $P(p_x, p_y) <- P(p_x, p_y) - c$

The problem with this algorithm is that the matrix $P$ loses an amount of probability $c$ for each iteration; hence, the probability must be shared by the neighborhood of the position $(p_x, p_y)$. If the probability were not shared, the matrix $P$ would no longer be a PDF satisfying the condition

$$\sum_{i,j} P(i,j) = 1$$

The probability of the neighboring values increases by an amount of $d = \frac{c}{4}$ or $e = \frac{c}{4\sqrt{2}}$ according to the distance from the values at the center in an 8 neighborhood. $4d + 4e = c$ is satisfied; therefore the function given by $P$ is still a PDF.

A new matrix of absorption ($A$) is required as a parameter for the algorithm to simulate the absorption of the paper. Each value of the matrix $A$ simulates a cell of paper, which means that the value of these cells is related to the maximal amount of ink that can be absorbed by the paper. When a cell of the paper does not admit more ink, the probability is shared with the neighbor cells, and the probability in the cell becomes 0. This condition can be verified using the matrix ($I$). The line can be added just before the parameter $c$ is subtracted from the cell to take into account the effect of viscosity:

3. IF $I(p_x, p_y) \leq A(p_x, p_y)$ THEN: $c <- P(p_x, p_y)$
4. IF $x_v < v$ THEN: $P(p_x, p_y) <- P(p_x, p_y) - c$

Increasing the value of the probability to $c$ shares the probability with the neighbor values in the algorithm of the previous subsection. Therefore, the algorithm of distribution is the same for the viscosity distribution and the absorption distribution. The basic algorithm is the following:

a) The seed is generated, $I$ and $A$ are not modified, the probability of the cells is updated in $P$.

b) The probabilities change according to the matrix of absorption and the viscosity of the ink, thus the probability of accumulating ink changes with every iteration.

c) The gray values are updated according to the given probability.

A value and its neighbors may have a probability of 0 while the gray value is not 1. This happens when the cells of the paper do not admit more ink. In this case, the criteria of expansion are 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 then the algorithm finishes.

### *4.3.4 Modifying the Algorithm to Generate Non-Uniform Dots*

The problem with the previous algorithm is that all the dots appear very regular due to the fact that the ink is propagated with the same probability. The effect produced is that the dot presents some irregularities due to the absorption of the paper and the viscosity of the ink, but when artists want to create a dynamic feeling in their works of art, they usually change the orientation of the pen and the time of exposition, modifying the dots in an arbitrary way. This is one of the characteristics that makes illustrations appear hand-drawn.

A new parameter, that we call the stress parameter, is introduced in order to try to simulate the speed and the inclination of the technical pen when artists stipple. It is given by an angle and a distance from the center of the matrix $I$. A value of 1 is the distance of the diagonal of this matrix. This stress parameter can also be defined as a position of the matrix, but it is usually easier to define masks in polar coordinates for interaction with the user. The Euclidean coordinates can be easily computed from polar coordinates. Finally, the distances to the stress point and to the initial seed are computed.

The algorithm that modifies the probabilities for every iteration is computed from these distances: $d_{sp}$ *[e] = d0 [e] + d1 [e]*, where *d0 [e]* is the distance from the seed, and *d1 [e]* is the distance from the stress point for every element ($e$) of the list *Lp*.

Once the distances have been computed for each element in the list, the algorithm obtains the probability $p_g$ of the cell with the greatest distance ($C_g$), and the probability $p_l$ of the cell with lowest distance ($C_l$). The probability of $C_g$ becomes 0, whereas the probability of $C_l$ is $p_g + p_l$. All the elements

with probability 0 are removed from the list $L_p$ at the end of the algorithm. The cells that are in between the initial seed and the stress point have greater probability.

The dots generated with this algorithm and the dots made by artists are compared in Figure 5. Dots have been compared statistically using a Structural Similarity Image Function (SSIM), revealing a 95% similarity [28].


Fig. 5: Side by side comparison of dots made by the artist (left) and generated by our algorithm (right).

### 4.4 User Interface and Tools Integration

We have designed and implemented a set of operators based on the process that artists usually follow in their illustrations. These operators are represented by tools that work in basically the same way: the user selects a tool and draws on a representation of the mask; finally, the operator changes the statistical function ($P_{pdf}$) and the output image ($I_{out}$). The user can enable or disable the visualization of the mask. Great precision is not required when drawing on this mask because the operators do not directly affect the function that places dots, but only the corresponding probabilities. An important point is that the user does not draw the illustration, but only marks the tasks that the application will carry out. The user decides when to stop the algorithm or when to continue with the stippling process, but he or she can make modifications while the automatic stippling is running. The user selects a specific tool mapped to an operator, where each operator corresponds to a particular technique or change of appearance in the illustration (such as size change of dots). The effects produced with these operators are shown and explained in Figure 6 and in the following paragraphs:

1) Remove: removes the values of marked values in $F_{pdf}$. Dots are then no longer placed in that area. $I_{out}$ is also deleted in that zone, so there will be no dots in the entire area.
2) Redraw: replaces the values in $F_{pdf}$, and removes the values in $I_{out}$. The algorithm places dots in that zone again.
3) Draw first: increases the value of marked values in $F_{pdf}$. The algorithm places more dots in that area. The values with probability less than or equal to 0 are not altered. $I_{out}$ is not altered.
4) Draw later: decreases the value of marked values in $F_{pdf}$. The algorithm places less dots in that area. The values with probability less than or equal to 0 are not altered. $I_{out}$ is not altered.
5) Enhance borders: changes the value of marked values in $F_{pdf}$ in such a way that the probability of the borders increases. It removes the values in $I_{out}$.
6) Shade: increases the probability in areas with less intensity; also increases contrast. $I_{out}$ is not altered.
7) Invert: changes $F_{pdf}$ and $F_{size}$ to simulate an inversion of the stippling by inverting the probability and adjusting the size of dots. $I_{out}$ is removed in that area.
8) Stop stippling: sets the values of marked values of $F_{pdf}$ to 0. The algorithm does not place a new dot in the image. $I_{out}$ is not altered.
9) Continue stippling: resets the values of marked values in $F_{pdf}$. $I_{out}$is not altered. The algorithm places dots in the area again.
10) Change the space of the dots: by changing the values of the surrounding values of $F_{pdf}$. The algorithm can prevent placing dots in the neighborhood of the values. $I_{out}$ is not altered.
11) Change size of the dots: the size of the dots can be easily modified by changing the values of $F_{size}$. $I_{out}$ is removed in the area.

These operators can be composited in such a way that the result simulates the described tasks in Sub-Section 3.1. Not all the operators can be composited, only those that do not remove the values of both functions ($F_{size}$ and $F_{pdf}$) at the same time. Conflicting operators cannot be composited either. Thus, the operators that can be composited are: Draw first, Draw later, Shade, Stop stippling, Continue Stippling, Change the space of the dots, and Change the size of the dots. For example, we can

combine an inversion operator (7) with a shade operator (6), and the result is an inversion, followed by an improvement of contrast details in the darker areas.

In the interface every operator is represented by a tool. Figure 6 shows how these tools work:

a) The user can display the default mask that corresponds to the statistical function (darker areas have more probability than lighter areas, the green values indicate a greater size for dots)

b) The user uses the Remove tool to eliminate the handle of the cup: the user paints the area to be removed in the mask, all other areas are stippled by the algorithm.

c) The user uses the Redraw tool on the handle, so the algorithm again stipples this area (note how this operator does not produce patterns; the transition of the dots is smooth).

d) The Draw-first or the Draw-later tools change the mask, darkening or lightening the areas.

e) The Enhance-borders tool has been used on almost the whole cup, removing the dots and changing the mask, with the result that the algorithm only stipples along the borders.

f) The Shade tool is applied outside of the cup, so the shading is greatly improved due to the use of a contrasted version of the image, as described in operation 6 (Shade), whereas the borders of the previous action are maintained.

g) The Continue/Stop stippling tool allows some select areas of the image to be stippled, the Continue stippling tool can also enhance the details of an area.

h) The size and the space of the dots can be changed with the appropriate tools.

i) The Brightness reversal tool (or inversion tool) allows a change in the way the stippling is produced. This tool has been applied to the whole image. It is worth observing how zones with probability equal to 0 are not stippled. This tool should be used in moderation, because in real illustrations artists only use inversion when too many dots would hide the details.

## 5 RESULTS

We have tested the quality of the results produced by our algorithms for dot generation and for dot placement, as can be seen in Figure 4 and Figure 5. The number of different dots is unlimited, and several parameters (such as the absorption of the paper) can be changed easily to obtain different kinds of dots, as shown Figure 7. In this figure viscosity and the color of the ink has been set to $\nu = 0.5$ and $k = 0.8$. The stress parameter has been set to 0 in a) and b), whereas this value has been randomly adjusted in c) and d). A screenshot of the interface is shown at Figure 8, which shows how the mask can be displayed and how it can be altered with the painting tool. The tools can be selected by using the menu on the right. The user paints over the mask with the pen, which affects the area given by the red circle (this circle can be resized). It also affects the probabilities, so the user does not need to draw on the mask with great precision. As shown in Figure 9, through the use of this user interface and of our software, realistic illustrations can be generated by non-artist. Most of the work is performed automatically by the tool. In Figure 9 the user simply chose where to enhance some borders or to intensify the shading.

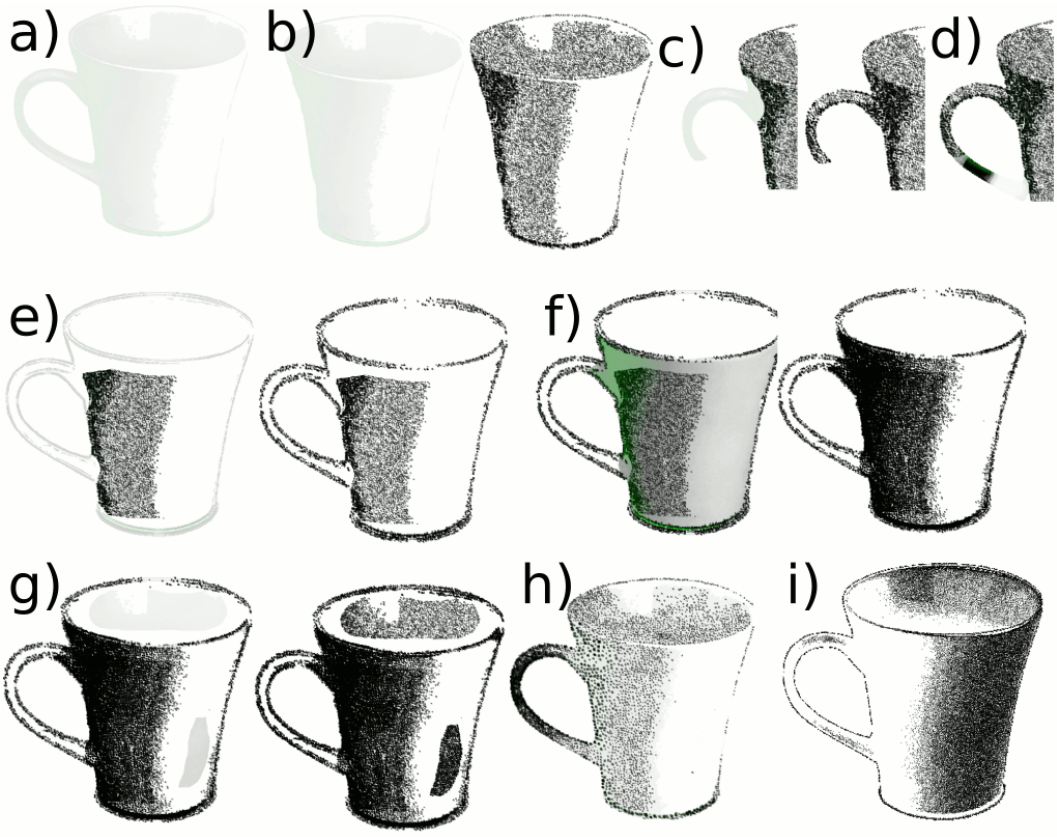


Fig. 6: An example of how the mask and the different tools of the user interface work.

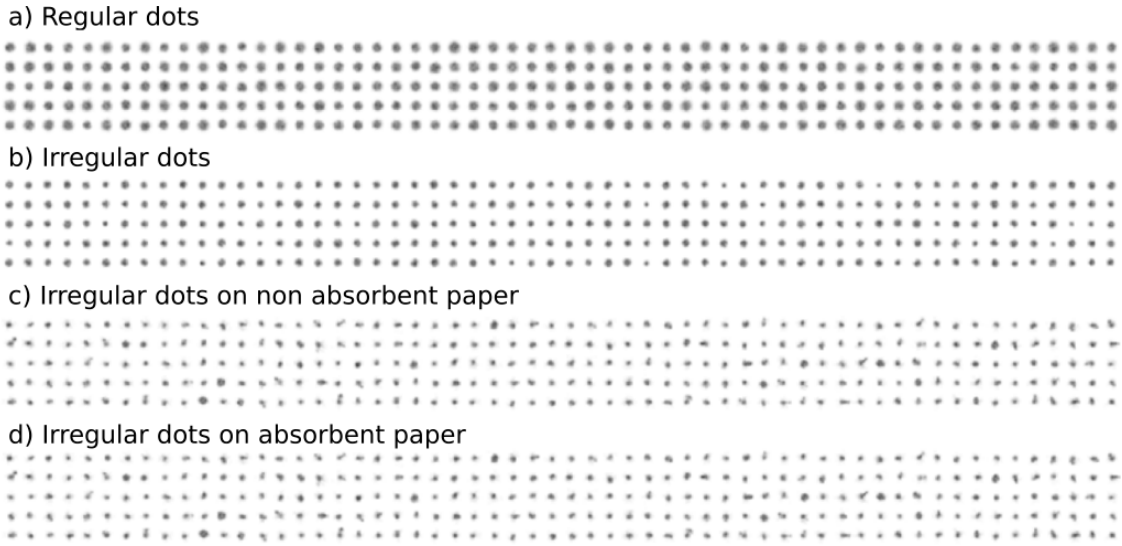


Fig. 7: An example of dot generation. The only parameters that have been modified are the absorption and the stress parameters (both described in Sub-Section 4.3).


Fig. 8: (Left) a screenshot of the main interface; it allows the user to select the size and color of the dots, the input photograph, and the absorption of the paper. (Right) a screenshot of the interface for automatic stippling; it shows the mask of the photograph as a background for placing the dots when the algorithm continues. The options on the right menu show the different tools for editing the stippling process (from top to bottom): Smaller dots, Larger dots, Closer dots, More distant dots, Continue stippling, Stop stippling, Invert, Shade, Enhance borders, Draw later, Draw first, Redraw, Remove.

## 6 CONCLUSIONS

In this paper, we have presented a method to generate dots for stippling. The obtained dots are almost identical to the dots generated by artists when they manually stipple illustrations. The presented algorithm uses an iterative stochastic method to generate the dots. This method uses an adaptable point distribution function that varies according to several adjustable parameters that describe the paper and the ink. The proposed algorithm can generate an unlimited number of different dots. We have presented a new algorithm to place these dots according to a statistical function without producing noticeable objectionable patterns. This function can be modified in an interactive manner. We have also presented a CAD tool that automatically generates stippled illustrations. This software can generate realistic-looking stippled illustrations with dots of different shapes and sizes. The user interface allows a novice user to modify any part of an illustration by selecting some of the common techniques used in traditional manual stippling and by painting on a simple mask that specifies in what areas the selected operation should be applied.

## ACKNOWLEDGMENTS

Authors thank the collaboration of the illustrator Elena Piñar who has drew the illustration in Figure 2 specifically for this work. Thanks to the Ministerio de Educación y Ciencia of Spain for the projects TIN2007-67474-C03-*02* and TIN2007-67474-C03-*01*, which have partially funded this work. Thanks to the editor for his useful comments that have helped to improve this paper.

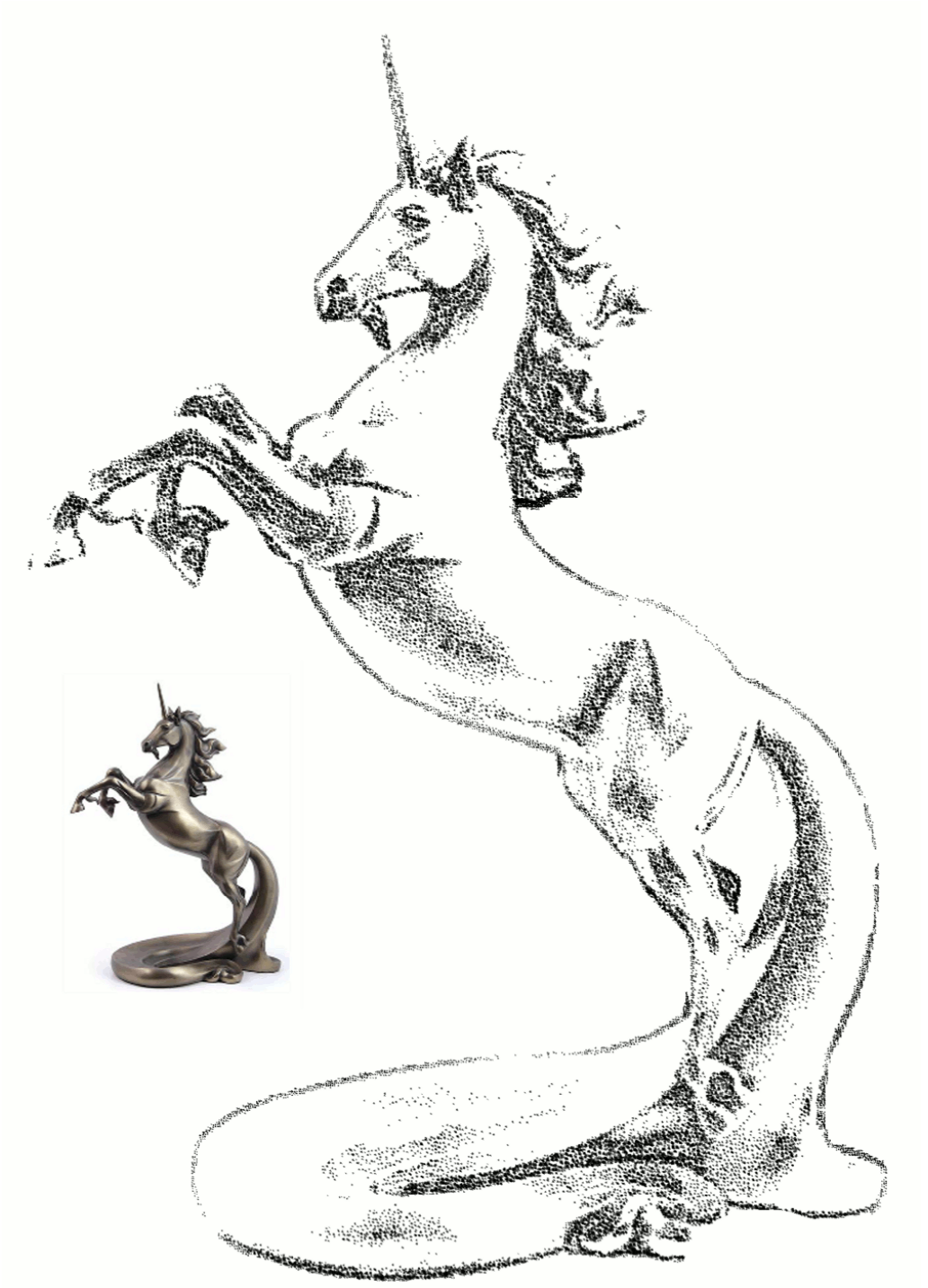

Fig. 9: Example of the output of our algorithms after only a few minutes of user interaction. Several tools have been used. Similarity to a manually stippled illustration is clear. The input photograph of the algorithm is shown on the left.

**REFERENCES**


[1] Ostromoukhov, V.: A Simple and Efficient Error-Diffusion Algorithm. In Proc. SIGGRAPH, ACM, New York, 2001, 567-572

[2] Hodges, E. R. S.: The Guild Handbook of Scientific Illustration, John Wiley and Sons, 1989.

[3] Pastor, O. M.; Freudenberg B.; Strohthotte, T.: Real-time animated stippling, In Proc. of NPAR 2004, 23, 2004, 62–68.

[4] Hoppe, H.: Progressive Meshes, In Proceedings of SIGGRAPH, August, 1996, 99-108.

[5] Lu, A.; Morris, C. J.; Ebert, D. S.; Rheingans, P.; Hansen, C.: In Proc. of VIS, IEEE Computer Society, Los Alamitos, 211-218.

[6] Lu, A.; Morris, C. J.; Taylor, J.; Ebert, D. S.; Hansen, C.; Rheingans, P.; Hartner, M.: Illustrative Interactive Stipple Rendering, IEEE Transactions on Visualization and Computer Graphics, 9(2), April-June, 2003, 127-138.

[7] Sousa, M. C.; Foster, K.; Wyvill, B.; Samavati, F.: Precise ink drawing of 3d models, In EUROGRAPHICS'03, Brunet, I. P., editors D.F., (Eds.), 22, 2003, 369-379.

[8] Xu, H.; Chen, B.: Stylized Rendering of 3D Scanned Real World Environments, In Proc. NPAR, ACM, New York, 2004, 25-34.

[9] Zakaria, N.; Seidel, H.-P.: Interactive Stylized Silhouette for Point-Sampled Geometry, In Proc. GRAPHITE, ACM, New York, 2004, 242-249.

[10] Yuan, X.; Nguyen, M. X.; Zhang, N.; Chen, B.: Stippling and silhouettes rendering in geometry-image space, In Proc. of Eurographics Symposium on Rendering, 193-200.

[11] Secord, A.: Weighted voronoi stippling, In Proc. of NPAR, ACM Press, 2002, 37-43.

[12] Deussen, O.; Hiller, S.; Overveld, C. V.; Strothotte, T.: Floating points: A method for computing stipple drawings, Computer Graphics Forum, 19, 2000, 40-51.

[13] Hiller, S.; Hellwig, H.; Deussen, O.: Beyond Stippling – Methods for Distributing Objects on the Plane, Computer Graphics Forum 22(3), 2003, 515-522.

[14] Dalal, K.; Klein, A. W.; Liu, Y.; Smith, K.: A Spectral Approach to NPR Packing, In Proc. of NPAR, ACM, New York, 2006, 71-78.

[15] Barla, P.; Breslav, S.; Markosian, L.; Thollot, J.: Interactive hatching and stippling by example, INRIA, 2006.

[16] Secord, A.; Heidrich, W.; Streit, L.: Fast primitive distribution for illustration, In Thirteenth Eurographics Workshop on Rendering, 2002, 215-226.

[17] Schlechtweg, S.; Germer, T.; Strothotte, T.: Renderbots: Multi agent systems for direct image generation, Computer Graphics Forum, 24, 2005, 283-290.

[18] Kopf, J.; Cohen-Or, D.; Deussen, O.; Lischinski, D.: Recursive Wang Tiles for Real-Time Blue Noise, ACM Transactions on Graphics 25(3), 2006, 509-518.

[19] Mould, D.: Stipple placement using distance in a weighted graph, In Proc. of Computational Aesthetics in Graphics, 3, 2007.

[20] Kim, D.; Son, M.; Lee, Y.; Kang, H.; Lee, S.: 2008, Feature Guided Image Stippling, Computer Graphics Forum, 27(4), 2008, 1209-1216.

[21] Kim, S.; Maciejewski, R.; Isenberg, T.; Andrews, W. M.; Chen, W.; Sousa, M. C.; Ebert, D. S.: Stippling By Example, In Proc. of NPAR, ACM, New York, 2009, 41-50.

[22] Isenberg, T.; Carpendale, M. S. T.; Sousa, M. C.: Breaking the Pixel Barrier, In Proc. of Cae, Eurographics Association, Aire-la-Ville, Switzerland, 2005, 41-48.

[23] Maciejewski, R.; Isenberg, T.; Andrews, W. M.; Ebert, D. S.; Sousa, M. C.: Aesthetics of hand-drawn vs. computer-generated stippling, In Proc. of Computational Aesthetics in Graphics, 3, 2007.

[24] Maciejewski, R.; Isenberg, T.; Andrews, W. M.; Ebert, D. S.; Sousa, M. C.; Wei, C.: Measuring Stipple Aesthetics in Hand-Drawn and Computer-Generated Images, IEEE Computer Graphics and Applications, 2008, 62-74.

[25] Abbot, M.; Basco, D.: Computational Fluid Dynamics - An Introduction for Engineers, Longman, 1989.

[26] Chu, N.; Tai, C.-L.: Real-time ink dispersion in absorbent paper, In Proc. of ACM SIGGRAPH, 24, 2005, 504-511.

[27] Jähne, B.; Scharr, H.; Körkel, S.: Principles of filter design, In Handbook of Computer Vision and Applications. Academic Press, 1999.
[28] Wang, Z.; Bovik, A. C.; Sheikh, H. R.; Simoncelli, E. P.: Image quality assessment: From error visibility to structural similarity, IEEE Transactions on Image Processing, 13(4), 2004, 600-612.