Chapter 3
Halftoning and Stippling

Oliver Deussen and Tobias Isenberg

3.1 Halftoning

Shortly after photography was invented, images became part of printed newspapers and books. William Fox Talbot, one of the inventors of photography, already mentioned an etching method (intaglio printing) for processing photographic screens which was commercialized in the 1880s.

Georg Meisenbach, a German inventor, developed and patented a halftone process on the basis of sets of parallel lines that were superimposed with the input photograph. He created his pattern by engraving lines in glass and darkened them using asphalt. Two or more such line patterns were superimposed and worked as a filter for the input image that divided it into dots of varying size (see Fig. 3.1).

3.1.1 Digital Halftoning

In digital halftoning, the screening process is implemented by representing the input image by electronically generated dots. Companies such as Linotype in the 1970s developed film recorders where the film was electronically illuminated dot by dot using precision optics.

O. Deussen (✉)
Dept. of Computer and Information Science, University of Konstanz, Konstanz, Germany
e-mail: oliver.deussen@uni-konstanz.de

T. Isenberg
INRIA Saclay, Orsay, France
e-mail: tobias@isenberg.cc

P. Rosin, J. Collomosse (eds.), *Image and Video-Based Artistic Stylisation*,
Computational Imaging and Vision 42, DOI 10.1007/978-1-4471-4519-6_3,

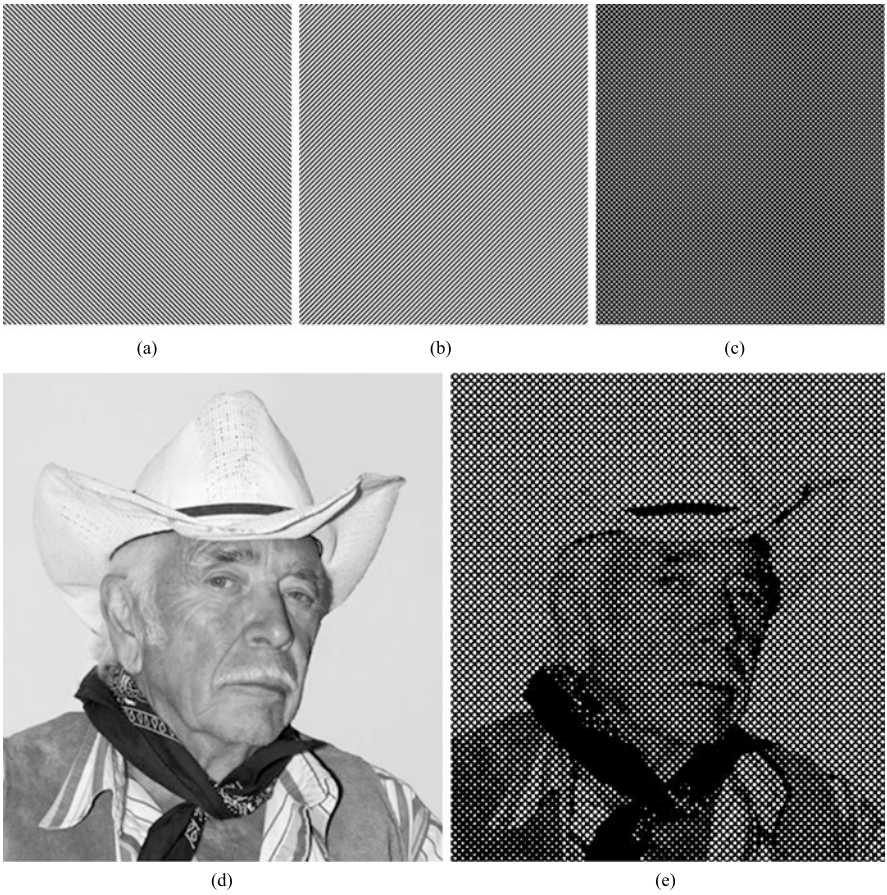


Fig. 3.1 Reproducing a photograph by multiple line patterns. (**a**) First line screen; (**b**) second line screen; (**c**) resulting screening pattern; (**d**) Input Image: Rosemary Ratcliff/FreeDigitalPhotos.net; (**e**) resulting halftoning pattern

When fully digital printing was invented for laser printers, halftoning meant to convert grayscale images into black and white pixels that were directly realized by printers.

3.1.2 Threshold Quantization

The easiest way to convert a given grayscale image into black and white pixels is to use a threshold, typically half of the highest intensity. Doing so, the resulting image shows large black and white areas (cf. Fig. 3.2(a)). A random variation of the threshold avoids such large areas but introduces noise (Fig. 3.2(b)).

(a) (b)

Fig. 3.2 (**a**) Quantization with fixed threshold; (**b**) threshold is varied randomly

Algorithm 3.1 One-dimensional Error Diffusion

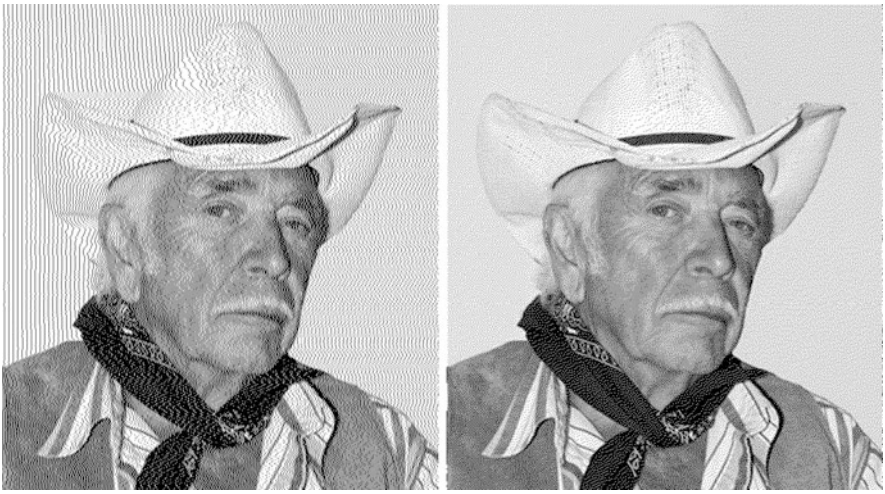
Input: A grayscale image
Output: An image of black and white pixels approximating the input

```
for y := height to 0 step 1 {
  for x := 0 to width−1 step 1 {
    if (input[x, y] > 127)
      K := 255;
    else
      K := 0;
    error := input[x, y] − K;
    input[x + 1, y] := input[x + 1, y] + error;
  }
}
```

Each decision if a pixel is represented in white or in black introduces a visual error. If the threshold is randomly varied, this error statistically averages and the overall image appears in the right tonal values. However, the noise remains and thus better techniques were developed.

Floyd and Steinberg [3] invented error diffusion for halftoning. When representing a pixel in white or black, the error in grayscale is determined and diffused to the neighbor pixels. The simplest case is one dimensional (cf. Algorithm 3.1), here a line of pixels is processed left to right and the error is diffused to the right neighbor.

In Fig. 3.3(a) the result of one-dimensional Floyd Steinberg quantization is shown. Vertical patterns appear since for every line almost the same error is distributed leading to almost the same patterns in every line. The two-dimensional



(a) (b)

Fig. 3.3 Floyd Steinberg quantization: (**a**) in the one-dimensional version vertical patterns appear; (**b**) the two-dimensional variant avoids this

Algorithm 3.2 Floyd Steinberg Error Diffusion

Input: A grayscale image
Output: An image of black and white pixels approximating the input

```
for y := height to 0 step 1 {
  for x := 0 to width−1 step 1 {
    if (input[x, y] > 127)
      K := 255;
    else
      K := 0;
    error := input[x, y] − K;
    input[x + 1, y] := input[x + 1, y] + 7/16 ∗ error;
    input[x − 1, y − 1] := input[x − 1, y − 1] + 3/16 ∗ error;
    input[x, y − 1] := input[x, y − 1] + 5/16 ∗ error;
    input[x + 1, y − 1] := input[x + 1, y − 1] + 1/16 ∗ error;
  }
}
```

variant of the algorithm (see Algorithm 3.2) avoids such patterns since here the error is not just distributed to the right neighbor but to the local neighborhood of the pixel.

Here four pixels around the current pixel are updated. The error is distributed with individual weights that were found by experiments. Figure 3.3(b) shows the improvement in the visual output.

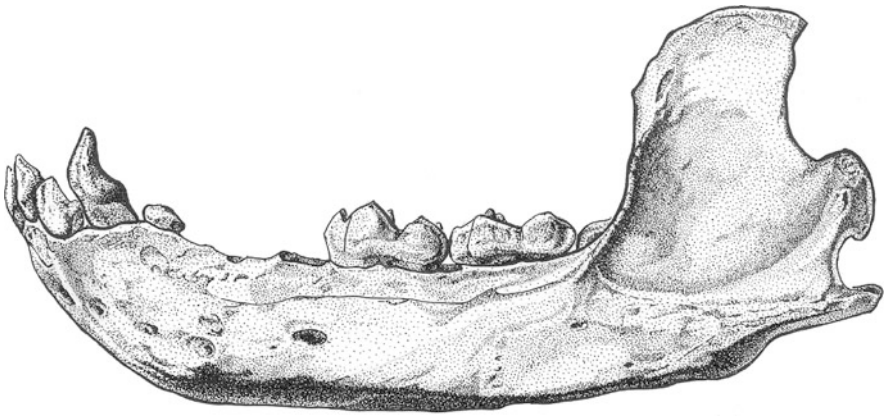


Fig. 3.4 Manually generated stipple drawing. Image courtesy by Brian L. Sidlauskas, Oregon State University

Many other halftoning techniques have been developed. Screening techniques imitate Meisenbach's superposition of an image with a repeating structure of varying width. A dither kernel (a small matrix of threshold values) that is repeatedly placed on the input image. Since the threshold values repeat regularly, the image impression is also more regular than with error diffusion.

3.2 Stippling

Stippling is an illustration technique that relies on dots. In contrast to halftoning methods, dots are distributed manually. The artists tries to set dots randomly but in most cases with almost uniform point-to-point distances.

The technique allows artists to represent tone plus material of an object, thus such techniques are used by scientific illustrators when objects have to be printed in black and white while faithfully representing their surface details. An example from biology is shown in Fig. 3.4, the surface of a bone is represented by the smooth arrangements of the dots, especially in the lighter areas almost uniform point-to-point distances are visible.

While in Fig. 3.4 silhouette lines were also used, in the following we want to concentrate on how to distribute points with the necessary characteristics by means of computers.

So called Centroidal Voronoi Tessellations create dot distributions that arrange points in the desired manner. Du et al. [2] describe them thoroughly, an analysis of different configurations of such tessellations and corresponding energy levels are given in [5, 16].

Centroidal Voronoi Tessellations for creating stipple patterns were introduced in [1, 17]. Other applications for such point sets are sampling [9] and numerical

integration [18, 19]. In both cases the spectral characteristics of such sets (Blue Noise characteristics) are the reason for their usage.

3.2.1 Voronoi Tessellations and Lloyd Relaxation

Let us assume that we have n points $S = s_1, \ldots, s_n$ on our paper. The Voronoi cell $V(s_i)$ of a point is the area around s_i for which each additional point on the paper would be closer to s_i than to any other point of the given set S. The regions of all points in S form a tessellation of the plane, meaning that they are pairwise distinct and jointly covering the entire plane.

Such tessellations are called the Voronoi diagrams $VD(S)$ of S. Since a tessellation of the entire plane would have open Voronoi regions for the outer points of S, typically such regions are closed by intersecting them with a (rectangular) frame that encloses the given point set. This frame is our canvas on which the points are distributed.

Ordinary Voronoi Tessellations use the Euclidean metric as a distance function. Many other distance functions can be used. However, for the purpose of Stippling the Euclidean distance is sufficient since it is a "natural" distance function with an intuitive relation between value and perceived distance.

Centroidal Voronoi Tessellations (*CVTs*) are ordinary Voronoi Tessellations with the additional property that every point s_i is placed in the centroid c_i of its Voronoi cell $V(s_i)$. The centroid (xc_i, yc_i) is defined by the moments of the area:

$$
\begin{aligned}
m_i^{0,0} &= \int_{y_{i1}}^{y_{i2}} \int_{x_{i1}(y)}^{x_{i2}(y)} dx\, dy \\
m_i^{1,0} &= \int_{y_{i1}}^{y_{i2}} \int_{x_{i1}(y)}^{x_{i2}(y)} x\, dx\, dy \qquad (3.1) \\
m_i^{0,1} &= \int_{y_{i1}}^{y_{i2}} y \quad \int_{x_{i1}(y)}^{x_{i2}(y)} dx\, dy
\end{aligned}
$$

$$
xc_i = \frac{m_i^{0,0}}{m_i^{1,0}}, \qquad yc_i = \frac{m_i^{0,0}}{m_i^{0,1}} \qquad (3.2)
$$

with A_i being the area of the Voronoi Cell and $x_{i1}, x_{i2}, y_{i1}, y_{i2}$ the boundary of the cell. Such a Cendroidal Voronoi Tessellation is shown in Fig. 3.5(b). The points are still almost at random but now with much less variance in their point to point distances.

Cendroidal Voronoi Tessellations can be achieved by applying the *Lloyd relaxation* to the points. This algorithm was invented by S. Lloyd in the 1960s at Bell Labs and later published in [11]. Each step of one iteration of this algorithm moves

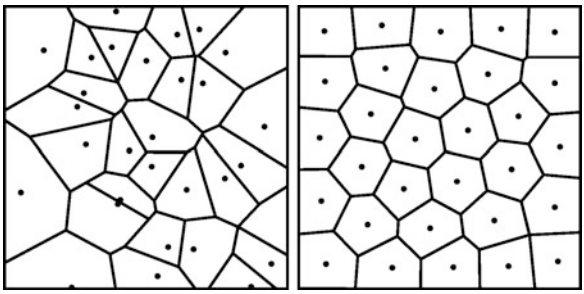


Fig. 3.5 (**a**) Voronoi Diagram; (**b**) movement of points during relaxation

each point towards the centroid of its Voronoi Region. If $c_i = (xc_i, yc_i)$ is the centroid of a Voronoi Region the movement can be represented

$$s_i^{(t+1)} = s_i^{(t)} + \alpha\left(c_i^{(t)} - s_i^{(t)}\right) \tag{3.3}$$

where $\alpha \in (0, 1]$ determines the speed of the movement. In the original algorithm α is set to one. The iteration is repeated until the movement of the points is below a given threshold. Figure 3.5(b) visualizes the movement during such a relaxation.

The CVT minimizes the following energy function, which measures the compactness of the Voronoi Regions (see [2]):

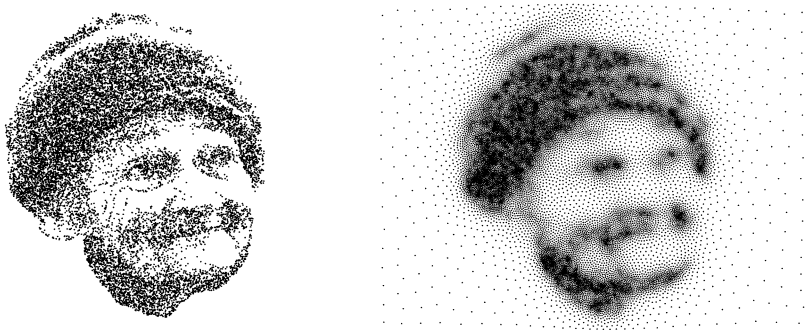
$$F_v\left(S, V(S)\right) = \sum_{i=1}^{n} \int_{V(S_i)} \|x - s_i\|^2 \, dx \tag{3.4}$$

This energy function sums up the integral of all quadratic distances of the points in a Voronoi Cell towards the corresponding point s_i. This is minimal for compact regions with almost uniform aspect ratio, thereby approximating hexagons [2]. Furthermore, it implies an almost uniform distribution of point-to-point distances since these distances are maximized for compact regions.

The Lloyd relaxation minimizes Eq. (3.4) and therefore can be used as a local optimization method for F. It moves the points into a distribution with almost uniform point-to-point distance. If continued for too long, it converges to a hexagonal distribution. This is not desired and typically iterating is stopped after a smaller number of steps.

3.2.2 Weighted Voronoi Tessellations

The Lloyd iteration distributes points in a uniform manner. However, for representing an input image we need varying density and thus have to find ways to modify the sizes of the Voronoi cells locally. Secord [17] published a variant of the iteration



(a) (b)

Fig. 3.6 (**a**) Initial Point distribution for a face; (**b**) point distribution after 20 steps of Lloyd's relaxation. The points spread over the entire drawing plane, image details are lost

in which every point on the plane is assigned a weight in proportion to the needed point density (grayscale value).

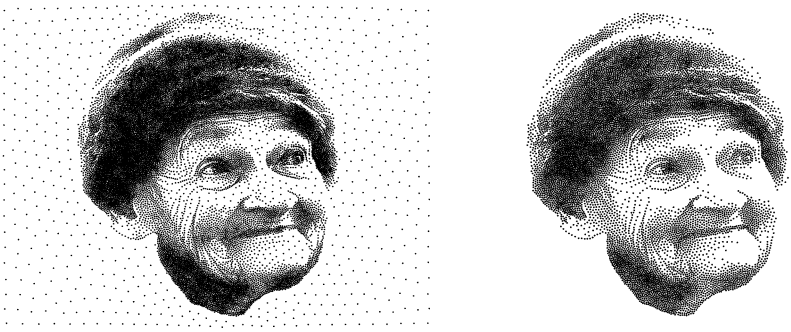
For doing so, the definition of the moments from Sect. 3.1 is modified in order to encapsulate the weights:

$$
\begin{aligned}
m_i^{0,0} &= \int_{y_{i1}}^{y_{i2}} \int_{x_{i1}(y)}^{x_{i2}(y)} \rho(x, y)\, dx\, dy \\
m_i^{1,0} &= \int_{y_{i1}}^{y_{i2}} \int_{x_{i1}(y)}^{x_{i2}(y)} x\rho(x, y)\, dx\, dy \\
m_i^{0,1} &= \int_{y_{i1}}^{y_{i2}} y \quad \int_{x_{i1}(y)}^{x_{i2}(y)} \rho(x, y)\, dx\, dy
\end{aligned}
\tag{3.5}
$$

with $\rho(x, y)$ being the density function that reflects the needed point density on the image. Parts of the equations can be precomputed, see [17] for details, and this allows for a fast computation of the integrals.

Let us have a look what can be produced with this simple optimization scheme. Since the method only moves points, an initial set has to be given. Usually one uses one of the above described halftoning methods and for every black pixel one dot is created.

Figure 3.6(a) shows such an initial point distribution for the face of a woman. If we apply the normal Lloyd Iteration for this point set, the points are spread over the entire drawing plane and all the details of the face are lost (cf. Fig. 3.6(b)). In Fig. 3.7(a) a weighted Centroidal Voronoi Tessellation is displayed. Here, in addition to the initial distribution the image itself was given to determine density and weights. Due to the weights the details of the face are captured better. If all points that are placed on white areas are removed the result looks similar to a stipple drawing (cf. Fig. 3.7(b)).



(a) (b)

Fig. 3.7 (**a**) Point distribution in weighted Voronoi Tessellation; (**b**) point distribution with points in white regions omitted

Fig. 3.8 Automatically stipple illustration of the bone from Fig. 3.4

Finally, in Fig. 3.8 the automatically stippled version of the bone of Fig. 3.4 is shown. Please note that this result is entirely automatic and no additional lines are inserted.

3.3 Beyond Stippling

In general, Lloyd's relaxation method can be applied to arbitrary objects, provided that their Voronoi Diagram can be calculated. For each object the center of gravity is determined. During iteration, the object is moved so that its center of gravity lies upon the center of gravity of its Voronoi Region. Additionally, objects can now be rotated (see Algorithm 3.3).

Algorithm 3.3 Modified Lloyd relaxation

Input: A set of objects o_i on the plane and a density function $\rho(x, y)$
Output: A relaxed centroidal Voronoi tessellation and a relaxed object-distribution.
1. Determine the mass centroids and main inertia axes of the objects o_i
repeat
2. Determine the Voronoi-Regions $V(o_i)$ of the o_i
for $i = 1$ **to** n **do begin**
3. Calculate the mass centroids (xc_i, yc_i) of the Voronoi cell $V(o_i)$
4. Move the mass centroids of the objects onto the mass centroids (xc_i, yc_i) of the Voronoi cells
5. Rotate the objects main axis so that it matches the main axis φ of $V(o_i)$
end
until the object positions converge

To compute the rotations we need to determine the second-order moments of the Voronoi cells:

$$
\begin{aligned}
m_i^{1,1} &= \int_{y_{i1}}^{y_{i2}} \int_{x_{i1}(y)}^{x_{i2}(y)} xy\,dx\,dy \\
m_i^{2,0} &= \int_{y_{i1}}^{y_{i2}} \int_{x_{i1}(y)}^{x_{i2}(y)} x^2\,dx\,dy \\
m_i^{0,2} &= \int_{y_{i1}}^{y_{i2}} y^2 \int_{x_{i1}(y)}^{x_{i2}(y)} dx\,dy
\end{aligned}
\tag{3.6}
$$

Using these moments and mass centroids one is able to calculate the main inertia axes and the desired rotation angle φ for the object. The two-dimensional inertia tensor is given as

$$
\mathbf{J} = \begin{pmatrix} \mu_{2,0} & \mu_{1,1} \\ \mu_{1,1} & \mu_{0,2} \end{pmatrix}
$$

The eigenvalues of $\mathbf{J}$ form the maximal and the minimal inertia moments j_1, j_2:

$$
j_{1,2} = \frac{1}{2}\left(\mu_{2,0} + \mu_{0,2} \pm \sqrt{(\mu_{2,0} - \mu_{0,2})^2 + 4\mu_{1,1}^2}\right) \tag{3.7}
$$

The angle φ of the main inertia axis is the angle of the eigenvector v_1 of $\mathbf{J}$ which belongs to the eigenvalue j_1:

$$
\varphi = \frac{1}{2}\arctan\left(\frac{2\mu_{1,1}}{\mu_{2,0} - \mu_{0,2}}\right) \tag{3.8}
$$

Figure 3.9 shows an object with its main axes (solid arrows), its Voronoi cell and its main axes (dashed arrows). These axes are determined for each iteration and the object is rotated so that the axes match.

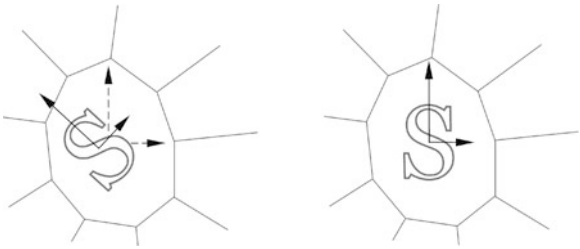


Fig. 3.9 Mapping the axes of the object (*solid arrows*) to the axes of the Voronoi cell (*dashed arrows*) defines the rotation of the object

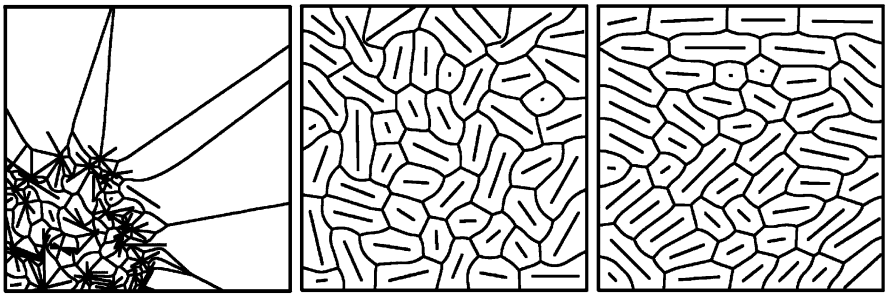


Fig. 3.10 A set of points and lines are relaxed using the extended relaxation method. © Eurographics Association, used by permission

This allows the user to set up the algorithm for the extension of Lloyd's relaxation that is able to incorporate object rotations. Within the process, it is assumed that first the mass centroid of the object is moved over the mass centroid of the corresponding Voronoi cell, and that the orientation is then adapted.

In Fig. 3.10(a) a set of points and lines is shown. Applying the method an even distribution as shown in Fig. 3.10(b) is achieved. Please note that the iteration works well with the very badly distributed initial set shown in Fig. 3.10(a).

The variants of the iteration can be mixed while working with an object set. Similar to what was proposed earlier [1], an interactive editor was built that allows the user to model sets of objects in various ways. One can move objects, insert or delete them using a number of "brushes". A special variant of the editor allows the user to apply one or more steps of each variant of the iteration.

The editor also enables the user to generate mosaics [4] (for more detail see Chap. 10). Here, small tiles have to be arranged in order to follow important structures in the input image, also the tile size was reduced for important regions such as the eyes to enhance precision. Examples are found in Figs. 3.11 and 3.12.

3.4 Stippling by Example

The techniques described so far create stipple patterns with a single characteristic—despite the fact that artists most often develop variants of such a patterns according

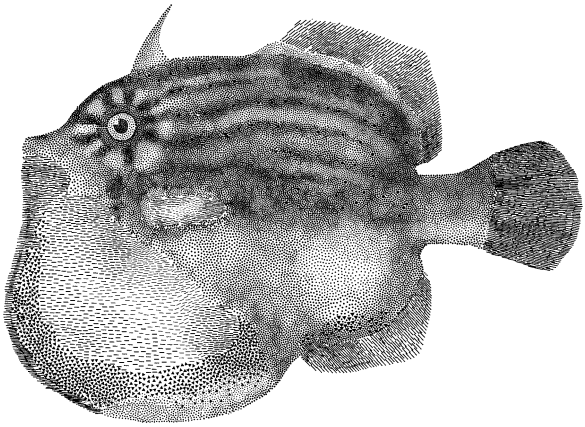


Fig. 3.11 A stipple illustration that uses different small objects as stipple marks. © Eurographics Association, used by permission

Fig. 3.12 Computer-assisted mosaics using the modified Lloyd's relaxation. © Eurographics Association, used by permission

to their own taste and style. It would, therefore, be desirable to create such stipple distribution patterns from examples given by the artists.

Such an approach was presented by Kim et al. [7]. For this work an example of a stipple drawing has to be given that incorporates different tonal values. In a first step patterns for a set of tonal values are extracted and their dot distribution is determined.

Since in hand-made stipple drawings the stipple marks have a variety of forms, such forms are extracted and stored together with the stipple distribution. A statistical analysis of the distribution [12] is used to find parameters for the stipple synthesis done on the basis of a texture synthesis algorithm.

Figure 3.13 shows example results of Kim et al.'s [7] approach. The grayscale image in Fig. 3.13(a) and the artistic stipple distribution in Fig. 3.13(b) is given

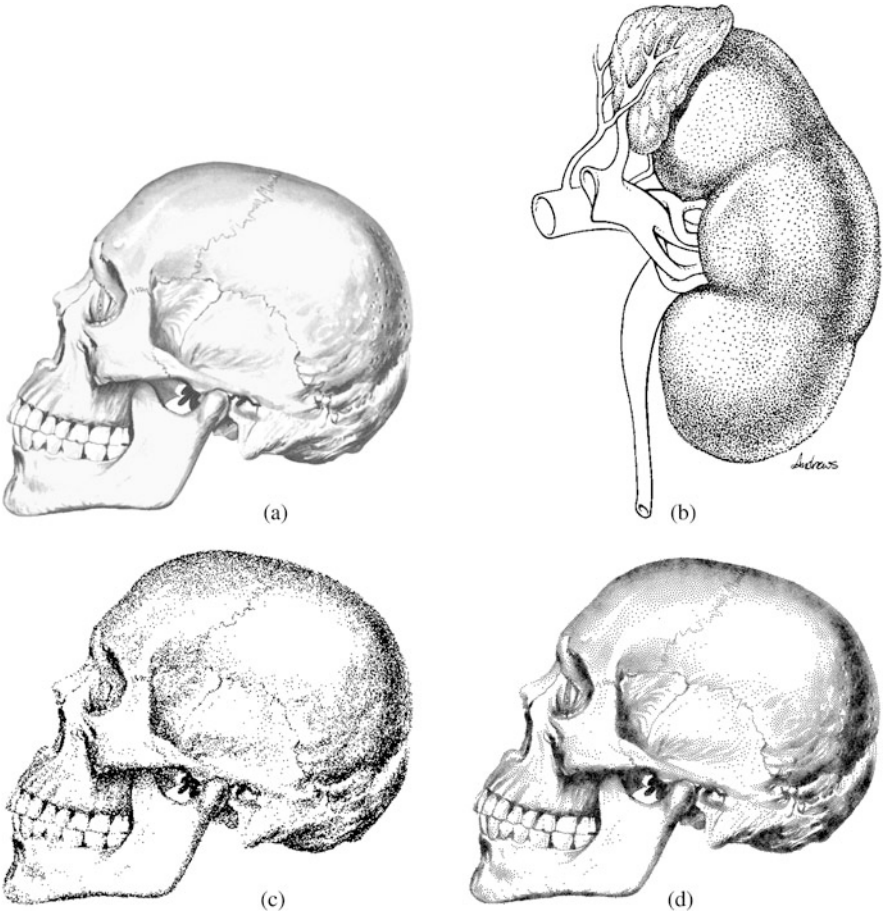


Fig. 3.13 Stippling by example: (**a**) grayscale image; (**b**) artistic stipple example used as the source of the example-based stipple distribution; (**c**) computer-generated pattern by example; (**d**) weighted Voronoi stippling for comparison. (**a**) Copyright © David Darling, used by permission, (**b**) copyright © 2009 William M. Andrews, used by permission, (**c**) copyright © 2009 Kim et al., used by permission

as input. The algorithm produces the new stipple drawing shown in Fig. 3.13(c). In contrast to the standard solution using weighted Voronoi Stippling (see Fig. 3.13(d)), the distribution of dots follows the characteristics of the input.

Another approach of stippling by example by Martín et al. [13, 14] does not look at the statistics of the stipple point distribution but, instead, at the resolution and scale at which the stipple dots are placed as well as the specific shape of the individual points. This technique addresses one of the limitations of the previous one by Kim et al. [7]—namely the inability of Kim et al.'s technique to produce the nice merging of stipple points in medium gray regions.

To address this issue, Martín et al. [13, 14] no longer treat the stipple points as black dots on a white background. Instead, they use grayscale scans of actual stipple points which they distribute based on halftoning. However, they also compute the correct resolution of stipple points on a paper of a given size, and use stipple dot scans of the appropriate pixel size so that the resulting scale-dependent images are produced for a given paper size (e.g., A4 or Letter paper). Moreover, due to the grayscale treatment, the grayscale stipple dots overlap and thus create the merging effects known from hand-made stippling. Interestingly, the randomized halftoning distributions that result from this process exhibit similar statistics as hand-made stippling, confirmed using Maciejewski et al.'s [12] distribution analysis.

Figure 3.14 shows an example of Martín et al.'s [13, 14] technique. Based on the input image in Fig. 3.14(a), a grayscale stipple image was produced at 600 ppi for A5 size (Fig. 3.14(b)), and the same stipple distribution was generated for the 1200 dpi black-and-white image in Fig. 3.14(c). Notice that due to the scale-dependence the same stipple dot scans and the same distribution is used for both images because they were produced for the same paper size—even though they have a different (pixel) resolution.

3.5 Structure-Aware Stippling

One issue raised, among others, also by Martín et al. [13, 14] is that low-level stipple placement is not sufficient for being able to produce high-quality illustrations. While Martín et al. analyze the stipple process by traditional stipple artist including such high-level processing, some authors have tried to incorporate higher-level processing into the NPR stippling pipeline (also compare some of the hatching techniques described in Chap. 4).

For example, Mould [15] transforms the input image into a regular graph whose edges are weighted according to the local gradient magnitude. Then, Mould uses a version of Dijkstra's algorithm to place stipple dots based on the graph. Next, Mould places the dots progressively along the frontiers of growing regions, which preserves the structure of meaningful artifacts such as edges in the images.

A related technique by Kim et al. [6] attempts to re-create the traditional hedcut illustration style which arranges stipples dots along lines for portraits of people. They first extract a line map and a tone map of the input image, then build a distance field and from that offset lines, and then use the extracted maps to optimize the placement of stipple dots. A different group of authors, Kim et al. [8], extend the initial portrait stipple approach by adding perceptual depth cues to the stipple image. Kim et al. [8] extract the image edges as well as a number of isophote lines (i.e., lines of the same brightness) and use these, similar to Kim et al. [6], as means to place stipple points. The isophote lines are extracted by quantizing the input image and extracting edges of identical color value. Based on the edge map and the isophote map, Kim et al. then produce a *weighted distance map* which they use to produce offset lines which, in turn, are used to place stipple points.

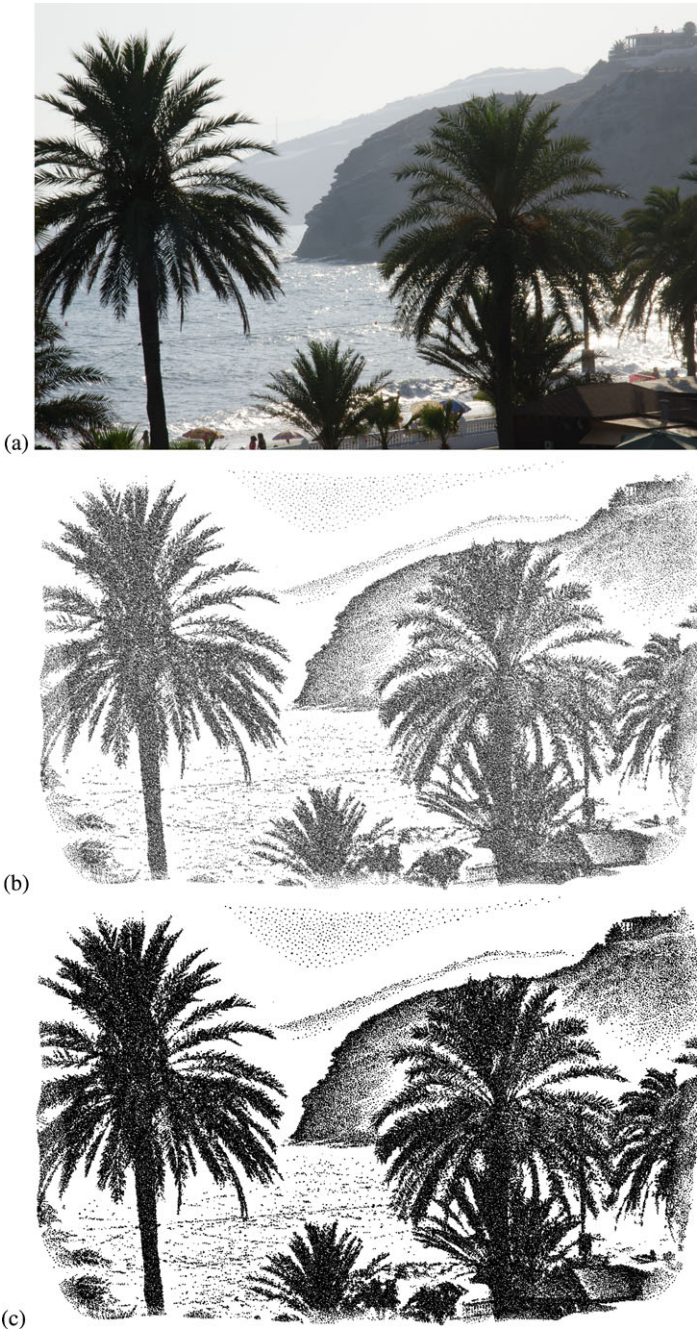


Fig. 3.14 Resolution-dependent stippling by example: (**a**) input image; (**b**) grayscale stipple result; and (**c**) black-and-white thresholded at a higher pixel resolution. (**a**) Copyright © 2012 Domingo Martín, used by permission; (**b**, **c**) copyright © 2012 Martín et al., used by permission

Recently, Li and Mould [10] suggested another way for stipple rendering to take the input image's structure into account using structure-aware error distribution that also permits the user to control how many stipple points are being used. For this purpose, Li and Mould consider one pixel to be one stipple and provide means to reduce the number of stipples by increasing the positive and reducing the negative errors to avoid having to place too many stipples (on a regular grid). Both types of error result in less pixels and, thus, less stipples being placed. This approach results in a high degree of preservation of the structure in the input image, even if only relatively few stipple dots are being used to represent the input image. Moreover, Li and Mould [10] also describe a number of further adjustments that result in different effects such as screening using textures or patterns.

3.6 Conclusion

Stippling is a powerful but cumbersome illustration technique for scientific illustrators since it allows to represent grayscale values and texture at the same time. It is widely used in archeology and biology. Computer-generated stipple drawings allow the production of such illustrations, for example for scientific visualization purposes, much faster since automatic methods exist that mimic what artists do.

Data-driven methods use patterns created by artists. They enable to capture the style of an illustrator quite precisely. Such methods enable the computer to create a much larger variety of patterns with different spatial characteristics; however, the price is that we have to use much more complex methods.

References

1. Deussen, O., Hiller, S., van Overveld, K., Strothotte, T.: Floating points: a method for computing stipple drawings. Comput. Graph. Forum **19**(4), 40–51 (2000). doi:10.1111/1467-8659.00396
2. Du, Q., Faber, V., Gunzburger, M.: Centroidal Voronoi tessellations. SIAM Rev. **41**(4), 637–676 (1999). doi:10.1137/S0036144599352836
3. Floyd, R., Steinberg, L.: An adaptive algorithm for spatial grey scale. Proc. Soc. Inf. Disp. **17**(2), 75–77 (1976)
4. Fritzsche, L.P., Hellwig, H., Hiller, S., Deussen, O.: Interactive design of authentic looking mosaics using Voronoi structures. In: Proc. 2nd International Symposium on Voronoi Diagrams in Science and Engineering 2005, pp. 1–11 (2005)
5. Gersho, A.: Asymptotically optimal block quantization. IEEE Trans. Inf. Theory **25**(4), 373–380 (1979). doi:10.1109/TIT.1979.1056067
6. Kim, D., Son, M., Lee, Y., Kang, H., Lee, S.: Feature-guided image stippling. Comput. Graph. Forum **27**(4), 1209–1216 (2008). doi:10.1111/j.1467-8659.2008.01259.x
7. Kim, S., Maciejewski, R., Isenberg, T., Andrews, W.M., Chen, W., Sousa, M.C., Ebert, D.S.: Stippling by example. In: Proc. NPAR, pp. 41–50. ACM, New York (2009). doi:10.1145/1572614.1572622
8. Kim, S., Woo, I., Maciejewski, R., Ebert, D.S.: Automated hedcut illustration using isophotes. In: Proc. Smart Graphics, pp. 172–183. Springer, Berlin (2010). doi:10.1007/978-3-642-13544-6_17

9. Kopf, J., Cohen-Or, D., Deussen, O., Lischinski, D.: Recursive Wang tiles for real-time blue noise. ACM Trans. Graph. **25**(3), 509–518 (2006). doi:10.1145/1141911.1141916
10. Li, H., Mould, D.: Structure-preserving stippling by priority-based error diffusion. In: Proc. Graphics Interface, pp. 127–134. Canadian Human-Computer Communications Society, School of Computer Science, University of Waterloo, Waterloo (2011)
11. Lloyd, S.P.: Least squares quantization in PCM. IEEE Trans. Inf. Theory **28**(2), 129–137 (1982). doi:10.1109/TIT.1982.1056489
12. Maciejewski, R., Isenberg, T., Andrews, W.M., Ebert, D.S., Sousa, M.C., Chen, W.: Measuring stipple aesthetics in hand-drawn and computer-generated images. IEEE Comput. Graph. Appl. **28**(2), 62–74 (2008). doi:10.1109/MCG.2008.35
13. Martín, D., Arroyo, G., Luzón, M.V., Isenberg, T.: Example-based stippling using a scale-dependent grayscale process. In: Proc. NPAR, pp. 51–61. ACM, New York (2010). doi:10.1145/1809939.1809946
14. Martín, D., Arroyo, G., Luzón, M.V., Isenberg, T.: Scale-dependent and example-based stippling. Comput. Graph. **35**(1), 160–174 (2011). doi:10.1016/j.cag.2010.11.006
15. Mould, D.: Stipple placement using distance in a weighted graph. In: Proc. CAe, pp. 45–52. Eurographics Association, Goslar (2007). doi:10.2312/COMPAESTH/COMPAESTH07/045-052
16. Newman, D.J.: The hexagon theorem. IEEE Trans. Inf. Theory **28**(2), 137–138 (1982). doi:10.1109/TIT.1982.1056492
17. Secord, A.: Weighted Voronoi stippling. In: Proc. NPAR, pp. 37–43. ACM, New York (2002). doi:10.1145/508530.508537
18. Smith, J.: Recent developments in numerical integration. J. Dyn. Syst. Meas. Control **96**(1), 61–70 (1974). doi:10.1115/1.3426777
19. Stoer, J., Bulirsch, R.: Introduction to Numerical Analysis. Springer, Berlin (1980)