UNDER REVIEW

RESCIENCE C

Editorial

Voronoi stippling

Ferdi Stoeltie

Edited by
(Editor)

Received
21 August 2022

Published
–

DOI
–

Introduction – In [1] by Adrian Secord a set of *techniques for generating stipple drawings from grayscale images using weighted centroidal Voronoi diagrams as in the traditional artistic technique of stippling that places small dots of ink onto paper such that their density give the impression of tone.*
This document describes a replication of the implementations in[1] by Adrian Secord. This paper describes a set of techniques for generating stipple drawings using weighted centroidal voronoi diagrams. It also describes several optimization techniques for better performance but a loss in detail. This document is part of a replication study wherein python code is written in - accordance with the original paper.

Methods – The original paper refers to OpenGL for gpu acceleration. Due to complexity reasons this has been left out of this implementation. Python has been used as the programming language. To generate the Voronoi diagrams, the Scipy Voronoi package has been used which uses the widely used Qhull library. The code has been written in a Jupyter notebook for easy access. As the Scipy Voronoi library does not deal with boundaries, code has been used that deals with this issue[2]. A special requirement in this implementation is that images have to be rectangular.

Voronoi diagrams – As the introduction suggests, the stipples are created by using voronoi diagrams. Thus it is important that voronoi diagrams are understood first. Imagine a two dimensional plane$p$. On this plane p there is a set ¶ containing points denoted as x. These points are defined by their x,y components on the plane.

Correspondence should be addressed to Ferdi Stoeltie (ferdi.stoeltie@student.hu.nl)
The authors have declared that no competing interests exist.
Open peer review is available at https://github.com/rescience-c/template.

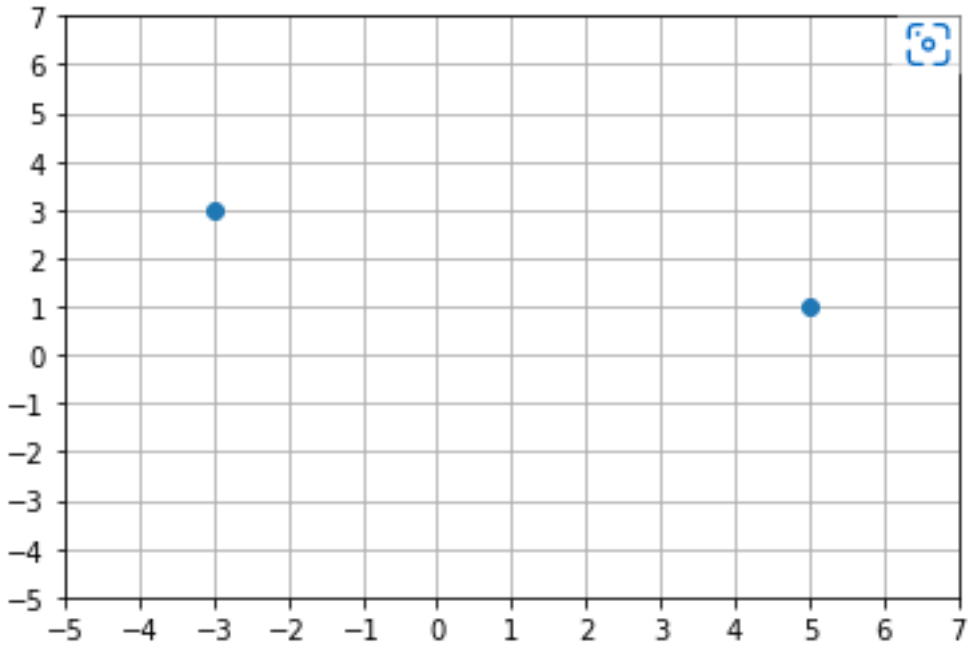


Figure 1. A top down view of plane p with points x_1, x_2. The plane can be infinite.

A voronoi diagram splits this plane into n regions A where n is equal to the amount of points x, so that all imaginary points in region A_n, are always closest to the 'real' point in region A_n. This is done by creating a line from point x1 to x2. Then on the midpoint of that line between x_1, x_2; create a new line perpendicular to the first. The new line divides all points on the plan into two regions A_1, A_2. See 2 for the result.
Creating a line perpendicular to line $\overline{x_1 x_2}$ is done by creating a two dimensional matrix containing the x,y components and then transforming them:

$$\begin{bmatrix} x_1 x & x_1 y \\ x_2 x & x_2 y \end{bmatrix} \rightarrow \begin{bmatrix} -x_1 y & x_1 x \\ -x_2 y & x_2 x \end{bmatrix}$$

The perpendicular line then only has to be set to the correct coordinates in the plane equal to the midsection of $\overline{x_1 x_2}$

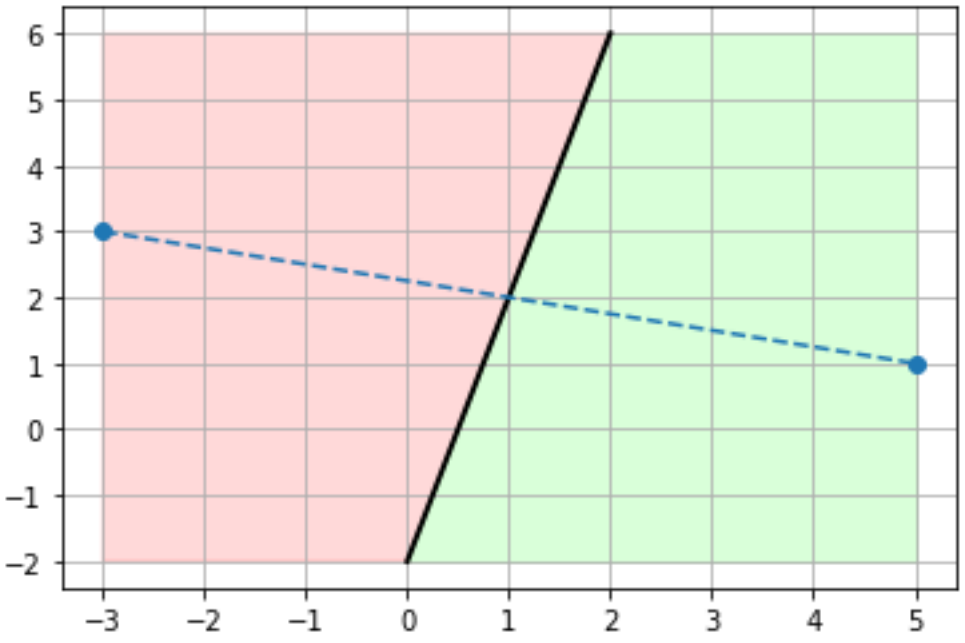


Figure 2. Dividing line separating the plane into two regions. Each of these regions defines an area wherein all points in that area are closer to either x_1 or x_2. All points on the line are equilateral distance from x_1, x_2

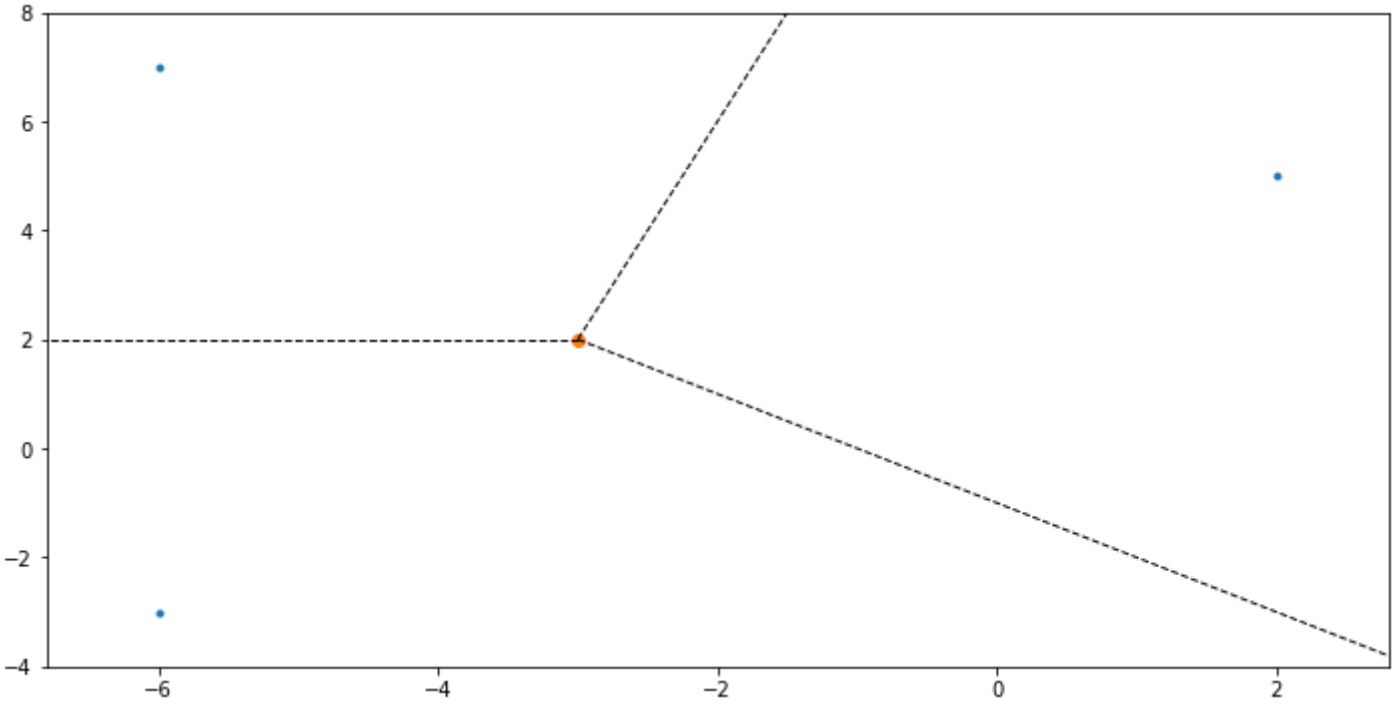


Figure 3. Example with three points in a Voronoi diagram.

Centroidal Voronoi diagrams — To go from voronoi diagrams to visually pleasing stippling images it is imperative to use some weighting function. A voronoi diagram splits a plane - such as an image - into n regions A. To determine where stipples should be placed a weighting function is used. The weighting function is applied to every region and is used to move the point inside each region to it's most average position based on. The weighting function takes the intensity of every pixel inside of a region and determines where the average intensity point in a region is. It then moves the real point in that region to the weighted function point. it does so for every 'real' point on the plane iteratively until all points are in their centroidal location within their regions.
The centroid of a region is defined as: $C_i = \frac{\int_A \boldsymbol{x} p(\boldsymbol{x}) dA)}{\int_A p(\boldsymbol{x}) dA}$
A is the region, $\boldsymbol{x}$ is the position and $p(\boldsymbol{x})$ is the density[1].

Generating Voronoi diagrams — Matplotlib has a voronoi module. Using this implementation has two major advantages:

- The code is fully functional and widely supported;
- The vertices belonging to a region are clockwise ordered, this is a requirement for the scan-line implementation later on.

Density function — Now that there is an understanding of how Centroidal Voronoi diagrams work, a function is required that generates the initial starting points using a density function. The density function will generate a map of points where darker regions will have more points and lighter regions fewer points. Rejection sampling was used to generate points over our intensity pixel distribution. All greyscale pixels are first converted into a scale (0 < v < 1) where 1 is white and 0 black. Then three vectors are

generated with n length using a uniform distribution function. The first two vectors are used to access pixel (x,y) data. The intensity of each pixel is then compared to the probability value. If it is greater than P(x) it is added to the list of starting regions.

Figure 4. Initial points from the density function using a uniform distribution

By using different distribution functions, it is possible to 'preserve detail' in some areas of the image. Currently a uniform distribution is being used but with a normal (gaussian) distribution the center of the image would be weighted heavier than the boundaries - resulting in more detail (stipples) in the center.

Generating stippling image — Using the result of the density function as our initial input. These are then used to iteratively generate centroidal voronoi diagrams until the average sum of the points update is less than some value. $\frac{\sum_{i=1}^{Gp} |old_i - new_i|}{Gp} < o$ where o is the convergence value and Gp the amount of points.
There is a distinct difference between the initial starting positions and after running the Voronoi algorithm until it has converged. In Figure 7a it is already clearly visible what the final output image will look like - just a lot grainier. This adds the benefit that options can be tweaked before running the Voronoi stippling to adjust things such as pixel intensity threshold[1], resolution of the image and amount of stipple points to generate.

[1]This threshold can be set to a level between $0 < intensity < 255$. The lower it is the more the algorithm will focus on darker regions

(a) Uniform distribution generation

(b) Centroidal Voronoi distribution

Figure 5. Difference between a hobbyist and an experienced professional

Precomputing stipple levels — The solution mentioned in section0.0.5 can be slow. A faster implementation is mentioned in the original paper[3]. By precomputing a range of stipple levels and applying these to the image a faster version can be created that computes a stipple image in $O(n)$ time where n is the amount pixels.

(a) Original (input) image

(b) Initial points

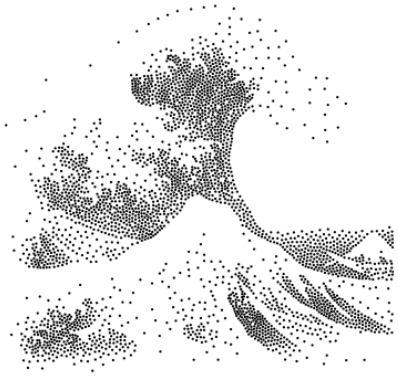
(c) Default Voronoi stipples generation

(d) Fast Voronoi stipples generation

Figure 6. Comparison between the default (slow) detailed implementation and fast implementation

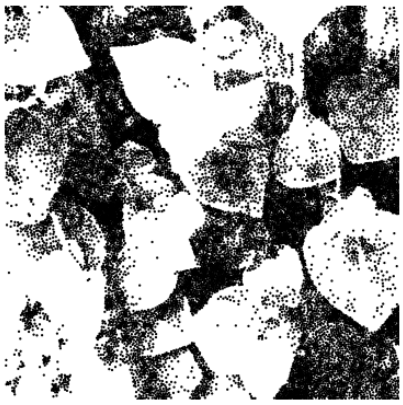
Results — The implementation yields similar visual results to those found in the original paper. Unfortunately no GPU with OpenGL support is available in this version.
The fast stippling method results in images similar in quality to the initial distribution7. The original paper does not offer a visualization of the initial distrubitions making it difficult to compare with the original results. Several configurations have been tried and tested for the fast voronoi stippling method but the results were quite similar.

(a) Original image (waves)



(b) Stippled waves (5000 stipples)

(c) Default Voronoi stipples generation



(d) Fast Voronoi stipples generation

Figure 7. Examples of stippled images

Conclusion — The resulting code offers similar results as shown in the original paper. The fast stippling method is assumed to be fast enough for real-time applications but unfortunately no test or comparison has been done. The results can be found in Github: https://github.com/FStoeltie/Vision

Recommendations — A few recommendations can be made for further study and improvements:

- Section Generating stippling image states that the results of the fast stippling method can be described as 'grainy' and comparable to the original distribution set. Developing a solution where these look more like the original Voronoi stippling method while keeping the complexity of $O(n)$ for real-time applications might be worth researching.

- The initial starting positions of points is the result of using an uniform distribution. By creating a custom function to generate points it is possible to add detail to certain parts - such as objects or contours. This might also result in more appealing images, especially for large and complex images multiple focal points.

References

1. A. Secord. "Weighted Voronoi Stippling." In: **Proceedings of the 2Nd International Symposium on Non-photorealistic Animation and Rendering**. NPAR '02. ACM, 2002, pp. 37–43.
2. A. FlabetVibes. "Getting a bounded polygon coordinates from Voronoi cells." In: **Getting a bounded polygon coordinates from Voronoi cells**. stackoverflow, 2016. URL: https://stackoverflow.com/questions/28665491/getting-a-bounded-polygon-coordinates-from-voronoi-cells.
3. M. Topalidou and N. P. Rougier. "[Re] Interaction between cognitive and motor cortico-basal ganglia loops during decision making: a computational study." In: **ReScience** 1.1 (2015). DOI: 10.5281/zenodo.27944.