

## Midterm Exam Report

Yunjoo Park

Advance Algorithm Programming

# 1 Summary of the two methods

A *Voronoi diagram* of a given point set  $P$  is the subdivision of the plane into  $n$  cells based on distance to points, one for each site of  $P$ , with the property that a point  $q$  lies in the cell corresponding to a site  $p_i$  if and only if  $\text{dist}(q, p_i) < \text{dist}(q, p_j)$  for each  $p_j \in P$  with  $j \neq i$ [1]. A *Centroidal Voronoi Tessellation* (CVT) is a special type of Voronoi diagrams. In the CVT, the point of each voronoi cell is the center of mass. Especially, a *Weighted Centroidal Voronoi Diagram* (WCVT) is a CVT in which each centroid is weighted according to a certain function. There are two methods to generate stipple drawings from images using weighted centroidal Voronoi diagrams. Both are based on the paper, "Weighted Voronoi Stipples"[?]. One is implemented by hedcutter. The other is implemted by voronoi method. Both generate well-spaced sets of points.

## 1.1 hedcuter method

This method is one way to generate stipple drawings from input image. This uses weighted centroidal voronoi diagram. Input is any image file and output is svg file. The program could have options below.

Parameter	Arg.	default	Description
-debug	X	false	Output with debugging information
-n	O	1000	Sample size
-uniform_radius	X	false	True, then all disks have the same radius
-radius	O	1	If uniform_disk is true, all disks have the disk size of the value. Otherwise, the largest disks will have the disk size of the value.
-iteration	O	100	Centroidal voronoid tessellation iteration limit
-maxD	O	1.01f	Max of site displacement
-black	X	false	True, then black disk

Table 1: Hedcuter method's options

This program uses Discrete Stippling algorithm[2] to stipple an image quickly.

---

**Algorithm 1** Discrete Stippling

---

```

for all pixel positions  $(x, y) \in [0, 1] \times [0, 1]$  do
    Map image value at  $(x, y)$  to stipple level  $l$ 
    Copy stipple on level  $l$  inside  $(x - \frac{1}{2}, y - \frac{1}{2}) \times (x + \frac{1}{2}, y + \frac{1}{2})$ 

```

---

In the program, build process undergoes several steps.

**First** Initialize sample points

**Second** Compute a Weighted Voronoi Tessellation

1. compute voronoi
2. move the site to the center of its coverage

**Third** Create disks

The method use 256 as the number required in a pure black image. We call the number  $N$ .

First of all, hedcutter generates sample initial  $n$  points. To be specific,  $n$  is initialized by *user*, or default value is 1000. For generating random points, it uses RNG (Random Number Generator). The random points are generated and distributed uniformly. The hedcutter calculates their value based on each color. If the point is close to black, the value is close to 0. On the contrary, if the point is close to white, the value is close to 1. This value is used to determines which points are appropriate. The points are decided whether to keep or not by comparing the value with a random number sampled from the gaussian distribution. Thus, the black has higher probability.

Second part is main of the hedcutter. At first, the hedcutter copies all points to *site of cells* and calculate *distance* for cells. The distance of each cell is obtained by converting its color intensity to distance value  $[0, 1]$  by equation (1). Let  $intensity(c)$  be the color intensity and  $distance(c)$  be the distance of the stipple.

$$distance(c) = \frac{(N - intensity(c)) * 1.0f}{N} \quad (1)$$

And then, sort cells using heap. The next step is about propagation. For each cell, the hedcutter calculate new distance of its neighbors. Neighbors are the points which are 3X3 matrix around the stipple.

$$distance_{new}(neighbor) = distance(c) + distance(neighbor) \quad (2)$$

If the calculated distance is less than the distance of the neighbor, push the neighbor to heap with the new distance. Finally, after the propagation step, the hedcutter collects cells by pushing points to coverage of corresponding cell and removes empty cells. For the collected cells, the hedcutter computes weighted average distance of each cell and updates the site of the cell to the new position using the distance. Repeat these process while satisfying the following conditions.

- the max of the displacement of site at each iteration < max\_site\_displacement (1.01f)
- the number of iterations < iteration\_limit (option: -iteration)

## 1.2 voronoi method

This method is also one way to generate stipple drawings from input image. This uses weighted centroidal voronoi diagram. Input file must be a PNG image. The program could have options below. Specifically, input and output are required[3].

Param.	Arg.	default	Description
-I	O		Must be a PNG file.
-O	O		The name of the file to output
-s	O	4000	The number of stipple points to render.
-c	X	OFF	Off, output in black and white
-t	O	0.1	The cut-off point for the convergence of the stippled output.
-n	X	OFF	Stipple points will not overlap.
-f	X	OFF	ON, then all stipple points have the equal size of radius.
-z	O	1.0	The radius of each stipple is multiplied by this value.
-p	O	5	Controls the tile size of centroid computations.
-l	X	OFF	ON, then produce an output log.

Table 2: Voronoi method's options

As I mentioned before, in CVT, each generating point lies exactly on the centroid of its voronoi region. The centroid of a region is defined as

$$C_i = \frac{\int_A x\rho(x), dA}{\int_A \rho(x), dA} \quad (3)$$

where  $A$  is the region of the cell,  $x$  is the position and  $\rho(x)$  is the density function. A centroidal voronoi diagram is a minimum-energy configuration in the sense that it minimize  $\int_A \rho(x)|C_i - x|^2$ .[?] This program uses Lloyd's algorithm[2] to generate a centroidal Voronoi diagram from any set of generating points.

---

### Algorithm 2 Lloyd's method

---

```

while generating points  $x_i$  do
    Compute the Voronoi diagram of  $x_i$ 
    Compute the centroids  $C_i$  using equation (3)
    Move each generating point  $x_i$  to its centroid  $C_i$ 

```

---

The algorithm find evenly spaced sets of points in subsets of Euclidean spaces, and partitions of these subsets into well-shaped and uniformly sized convex cells. That is, Under the algorithm, a voronoi diagram become a centroidal voronoi diagram through relaxation step.

The program consists of the following several steps.

**First** Create stippler

**Second** Loop while (average displacement > threshold)

1. distribute stipbles
  - (a) create voronoi diagram
  - (b) redistribute stipbles

**Third** Render

In the first step, creates initial stipbles. The second step is the main part. Firstly, in the *createVoronoiDiagram()* part, generates a voronoi diagram using stipbles. Second, in the *redistributeStipples()* part, calcuates the centroid of a cell using edges of the cell. To be specific, in the process of calcuating the centroid, creates and computes clip lines. Each clip line is extension of each edge. Thus, the clip lines form a polygon. Tests whether each point around the stipple is outside of the polygon or not. gets the density from image for inside points, calculates the density of the cell, and returns the center and radius of the cell. Updates stipbles of voronoi diagram with the centroid. After updating, calculates the average displacement of stipbles, denoted by  $t$ . Repeat this second step while  $t$  is greater than the *threshold*.

## 2 Comparison of the two methods

### 2.1 Do you get the same results by running the same program on the same image multiple times?

1. Hedcuter

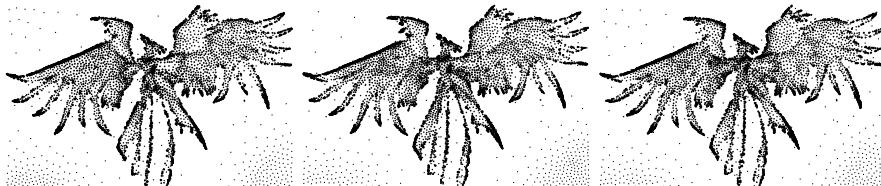


Figure 1: phoenix with 5000

2. Voronoi



Figure 2: phoenix with 5000

While I got the different outputs by running the hedcuter program on the same image multiple times, I obtained the same outputs by running the voronoi program. The hedcuter program generates sample initial points randomly and make a CVT under limit iteration and given max site displacement. Since the initial sample points set are different, the results of propagation are also changed. On the other hand, the voronoi program has the same initial points and repeats the series of process until the average of displacement of points is small enough. Thus, we could get the same output multiple times with the voronoi program.

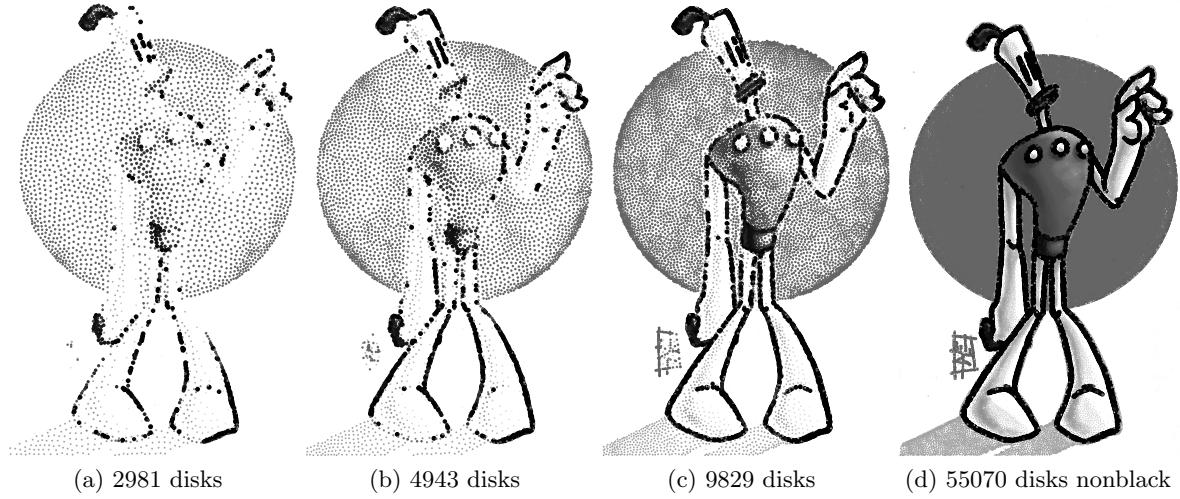


Figure 3: Hedcuter klaymen

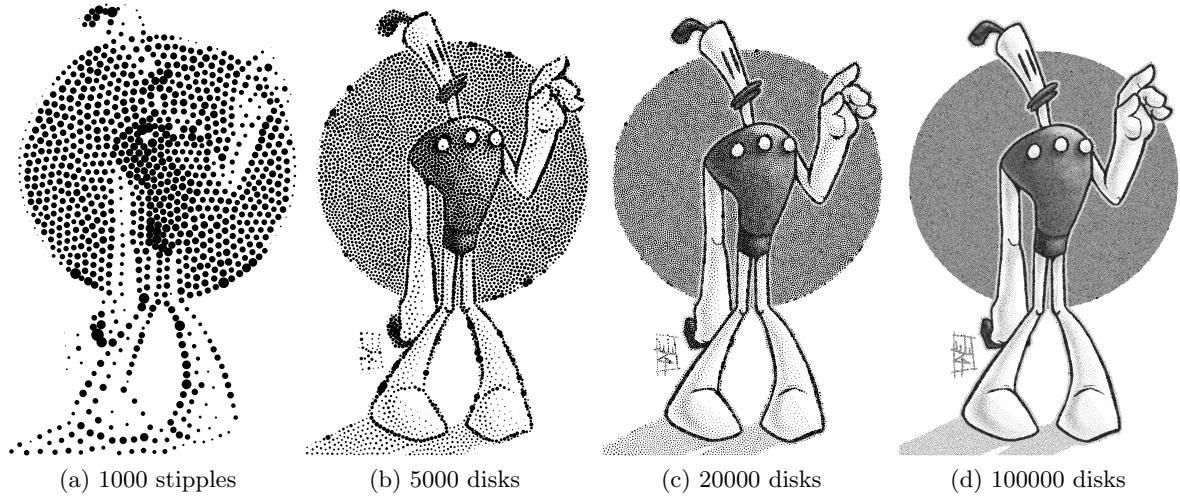


Figure 4: Voronoi klaymen

## 2.2 If you vary the number of the disks in the output images, do these implementations produce the same distribution in the final image? If not, why?

Both methods produce the different distribution in the final image when the the number of the disks in the output images varies. As the number of the disks increases, they generate the more detailed results. See the figure 3, 4

The hedcuter produces the different distribution the number of the disks in the output image,

## 2.3 If you vary the number of the disks in the output images, is a method faster than the other?

I could not assure that one method is faster than the other. In the hedcuter method, there are some points which oscillate after a few iteration in some cases. Thus, if there is no iteration limit, it will take infinite time because some points will oscillate infinitely. In the voronoi method, the more I increase the number of points, it takes the small number of iteration. Figure 5 shows elapsed time of the voronoi as the number of stipPLES is varied. As the number of points increases, the method becomes faster. However, the elapsed time increases again when the number of points goes beyond a certain level. This is because it takes too much time for each iteration even though there are few iteration.

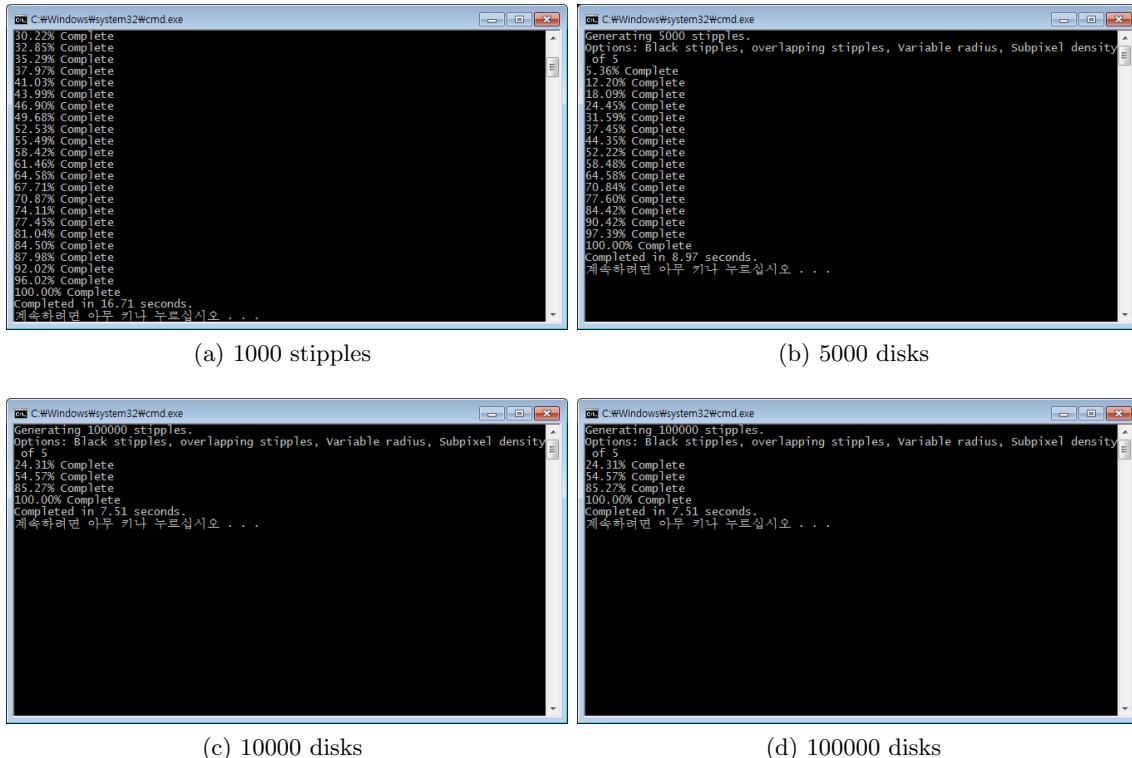


Figure 5: Elapsed time of Voronoi as a function of the number of stipPLES

- 2.4 Does the size (number of pixels), image brightness or contrast of image increase or decrease their difference?
- 2.5 Does the type of image (human vs. machine, natural vs. urban landscapes, photo vs.painting, etc) increase or decrease their difference?
- 2.6 Are the outputs of these stippling methods different the hedcut images created by artists (e.g. those from the Wall Street Journal)?

### 3 Improvement of hedcuter method

### References

- [1] Mark Overmars Otfried Cheong Schwarzkopf Mark de Berg, Marc van Kreveld. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2008. [1](#)
- [2] Adrian Secord. Weighted voronoi stippling. In *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*. [1.1](#), [1.2](#)
- [3] Sahab Yazdani. saliences.com: weighted voronoi stippling. [1.2](#)