Project Report for CS461

Single Line Drawing

Aman Raj
`aman170101006@iitg.ac.in`

Mayank Wadhwani
`mayan170101038@iitg.ac.in`

November 13, 2020

Abstract

The traditional technique of drawing single line images required use of machine hardware or experiences artists. We present a software-based solution to this problem by making use of concepts like Voronoi Diagrams, Bézier curves, stippling. An iterative technique acts on input images directly to produce high-quality single line drawings after stippling the input.

1 Introduction

This is the era of filters, we see so many different types of filters around us, people love it when they see different types of outputs on their faces. So we have attempted to construct our own small filter which can be used in different softwares such as Snapchat and Instagram. Moreover, Single Hand Drawing was a famous art form practised in the early 18th century, and it took weeks for the artist to construct the same. With the advent of technology, this bridge can be lowered to a couple of seconds where given a portrait of a person as an input, our software would consturct an equivalent image in a blink of an eye.

We have taken references from the paper *"Weighted Voronoi Stippling"*. to generate stipple drawings from images with as little user input as possible. The goal is to develop a tool which can generate high-quality stipple drawings from any source whatsoever, which implies that we use images as input and not 3D models. While this limits the amount of information we have to work with, it allows us a greater variety of input sources. For example, a user could start from a scanned pencil sketch, a photograph, the output of a 3D interactive application, frames of an animation, etc. The heart of the paper uses centroidal Voronoi Diagrams to compute the outputs. After generating the stippled images, we have made use of some techniques given in the paper *A Flexible Low-Cost Approach for the Generation of Physical Monochrome Art.* to construct single hand drawings using straight lines and Bezier curves respectively.

We have first provided a small background which is required to understand what is being done in the background of our software. Namely, we briefly describe the Voronoi Diagrams and Bezier Curves. We then provide the main methodology our software, the workflow and the algorithms used follows. As rightly said, *A picture is worth a thousand words*, we have also supplemented the use of our tool with some interesting results that we have achieved. We then talk about our efforts to build the project and finally discuss about some limitations of the project along with the future work that is possible to improve the user experience and efficiency of the single hand drawings.

2 Background

The following sections contain a couple of concepts to set the context of the working our algorithm.

2.1 Voronoi Diagrams

An ordinary Voronoi diagram is formed by a set of points in the plane called the generators or generating points. Every point in the plane is identified with the generator which is closest to it by some metric. The common choice is to use the Euclidean L distance 2 metric

$$|x1 - x2| = \sqrt{(x1 - x2)^2 + (y1 - y2)^2} \tag{1}$$

The set of points in the plane identified with a particular generator form that generator's Voronoi region, and the set of Voronoi regions covers the entire plane.

A centroidal Voronoi diagram has the interesting property that each generating point lies exactly on the centroid of its Voronoi region.

$$Ci = \int (x\alpha(x)) / \int (\alpha(x)) \tag{2}$$

where α is the density function of the voronoi region and x is the corresponding x coordinate.

2.2 Bézier Curves

Bézier curve is a parametric curve used in computer graphics and related fields to draw shapes, for CSS animation and many other purposes. Bézier curves are used to model smooth curves that can be scaled indefinitely. They are also used in the time domain, particularly in animation, user interface design and smoothing cursor trajectory in eye gaze controlled interfaces.

A Bézier curve is defined by a set of control points P_0 through P_n, where n is called its order (n = 1 for linear, 2 for quadratic, 3 for cubic, etc). The first and last control points are always the end points of the curve, however, the intermediate control points generally do not lie on the curve. Hence, they can be used as an approximation to connect various points using a smooth line.

Cubic Bézier curves are defined by 4 control points P_0, P_1, P_2 and P_3 in the plane. The curve starts at P_0 going toward P_1 and arrives at P_3 coming from the direction of P_2. Parametric equation of the curve is given as:

$$B(t) = (1 - t)^3 P_0 + 3(1 - t)^2 t P_1 + 3(1 - t)t^2 P_2 + t^3 P_3, 0 \leq t \leq 1 \tag{3}$$

3 Methodology

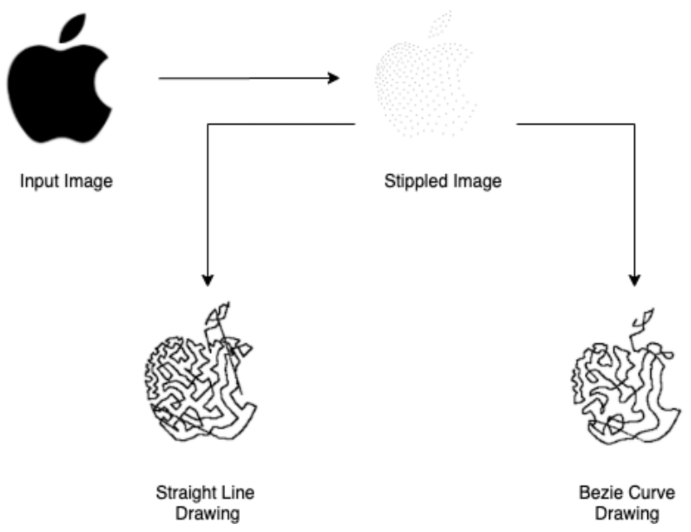


Figure 1: Project Workflow

3.1 Stippling

To perform stippling, we have used the methodology as explained in The Weighted Voronoi Stipping paper, namely we have used the Lloyd's algorithm as stated below:

Algorithm 1: Lloyd's Method

```
1 Initialization;
2 while generating points not converged to centroids do
3     Compute the Voronoi Diagram of x;
4     Compute the Centroids Ci using equation 1;
5     Move each generating point xi to its centroid Ci
6 end
```

We begin our algorithm by first computing the Voronoi Diagrams for our given set of generating points. These generating points are taken to be random initially. We run a multi-source breadth search traversal algorithm to find the nearest generating point for each pixel in the image. This has a time complexity of $O(n^2)$.

We then for each Voronoi region, calculate its centroid by summing the values of $\alpha(x)$ for that region. This step also taken $O(n^2)$ time complexity. After finding the numerator and denominator, we finally divide them to get the centroids for each Voronoi region in our image. We then equate our initial generating points to these centroids. We finally stop when the sum of Euclidean distances between the generating points and the centroids fall below a certain threshhold which can be taken as a point of convergence in the practical scenario.

3.2 Single Line Drawing

Line Drawing is the shape encoding technique that we have used to give shape to our stippled image obtained as output of previous section. We will be using a single line to connect all those stippled points using OpenGL in Python. Following is the algorithm to find a path connecting all the points.

Algorithm 2: Algorithm to find path

```
1 Start with a random point.;
2 while any non-visited point exists do
3     Find k-nearest points of current point;
4     Choose one out of them randomly;
5     Add it to the path;
6     Mark it visited;
7     Update current point to that point;
8 end
```

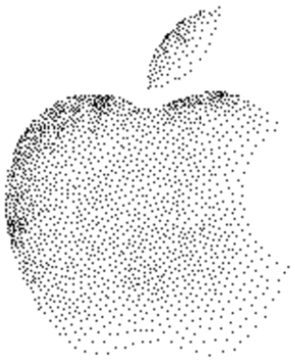
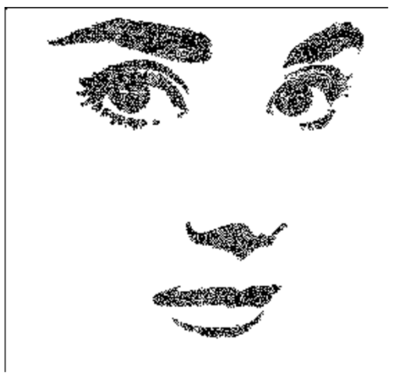
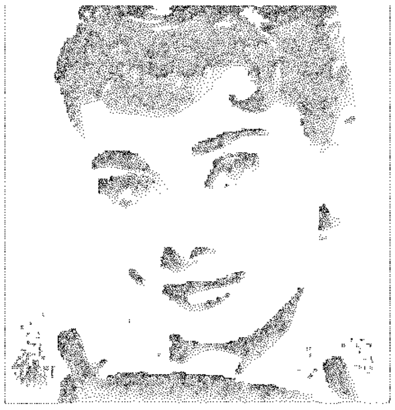
After we have a path/order of points in which we want to connect the points, we iterate through all the points in the path and connect them by straight lines or by bezier curves.

1. For joining using straight lines, we take all the points in path and iterate in pairs and simply use inbuilt OpenGL library function or simply add numerous points in between those two points using a straight line equation.

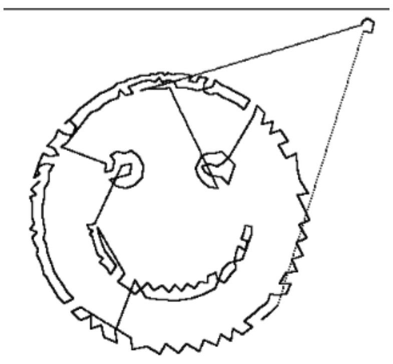
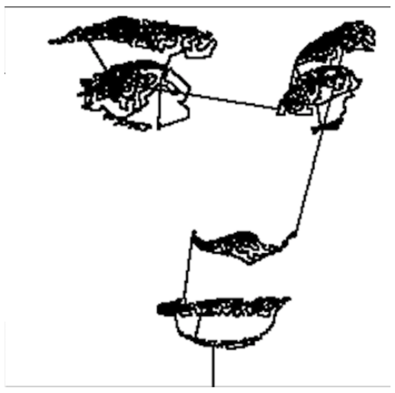
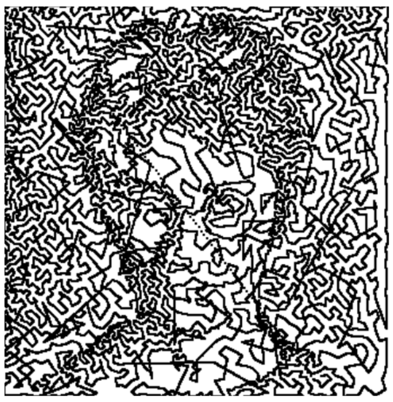
2. For joining using bezier cures, we take all the points in path and iterate in group of 4 and draw a bezier using all those points as control points for the parametric curve. This produces a smooth free hand looking single line drawing for the image.

4 Results

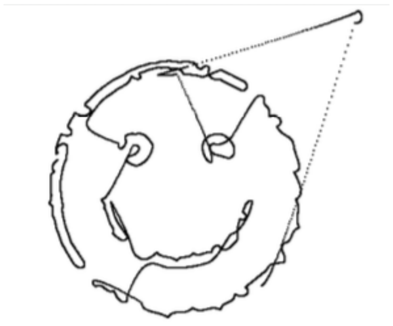
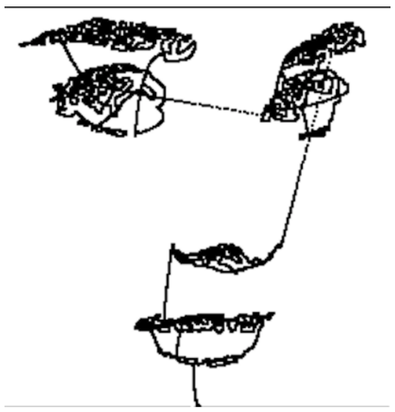
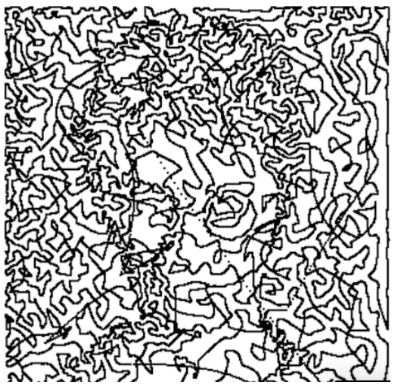
4.1 Stippling

4.2 Single Line Drawing using Straight Lines

4.3 Single Line Drawing using Bezier Curves

5 Our Efforts

5.1 Stippling

Even though there were available resources to perform stippling, we decided to implement everything from scratch to improve our comprehension of the text. We first began by studying the complete research paper and discussed on the various approaches we could have taken in our algorithm. Even though the author has suggested some other methods to compute the Voronoi Regions and centroids, we decided to implement the same using simple concepts and algorithms like Breadth First Search Traversal. Since we were unfamiliar with the libraries like PIL, we spent quite some time in understanding the same.

5.2 Single Line Drawing

First, we implemented the algorithm to find a path to draw the single line passing through all the stippled points. Then, we wrote the code to connect all the points in the path by drawing straight lines using OpenGL in Python. After that, we implemented the code to draw a single bezier curve to pass through all the points, by drawing multiple continuous bezier curves and ensuring all types of continuity on all the connecting points, taking help from the written assignment. But, it didn't produce satisfactory results. So, we compromised a little and changed our code to connect all the stippled points as control points for multiple bezier curves, which doesn't touch all the points but produces an approximation for the path.

6 Limitations and Future Work

1. The time complexity for our algorithm is $o(n^2)$ which may take some time to compute for very large images consisting of a large number of pixels. In that case, we may use some optimization techniques as given in the more recent works on stippling.

2. The final output generated by Bezier curves were just piecewise smooth and not smooth entirely. This was done by making batches of pixels and operating on them individually instead of working on the entire batch as a whole. This was done so that in the future, we may use the other cores and perform parallel computing to get the results faster. We may use clustering algorithms to detect clusters of regions which would have improved our efficiency of the image.

3. We can also make use of contour detection algorithms to detect and find boundaries of the image and then change the width of these boundaries, so given a portrait, we will be able to draw the face with a thick line and the interiors with a thinner

line giving rise to a better portrait.

4. We can also add colour to the output of the image. We can check the distance between two points and if it crosses a certain threshold, we can assume that the region has changed since similar regions will have many closely packed stipples. In this fashion, we will be able to add different colours to the image. We can make use of the above clustering and boundary detection algorithms for colour also. So different clusters or boundaries will have different colours.

References

[1] Adrian Secord. *Weighted Voronoi Stippling.*. 2002.

[2] Sergej Stoppel. *LinesLab: A Flexible Low-Cost Approach for the Generation of Physical Monochrome Art.*. 2019.