A Technical Report on

Design of a Laser Engraving System

**Submitted by**

Name: A K M Sharif Kaiser

Address: Bergrat-Mahr-Str. 18, 98693 Ilmenau, Germany

Phone: +4915123297506

E-mail: a-k-m-sharif.kaiser@tu-ilmenau.de

Matriculation Nr. 59728

Programme: Research in Computer & Systems Engineering (MSc.)

Technische Universität Ilmenau, Germany

**Supervised by**

Dr.-Ing. Detlef Streitferdt

Technische Universität Ilmenau

Fakultät für Informatik und Automatisierung

Institut für Technische Informatik und Ingenieurinformatik

Fachgebiet Softwarearchitekturen und Produktlinien

Postfach 100565

D - 98684 Ilmenau/Thüringen

E-Mail: detlef.streitferdt@tu-ilmenau.de

**Table of Contents**

[1. Project Overview 3](#_Toc68443279)

[1.1 Objective 3](#_Toc68443280)

[1.2 Background 3](#_Toc68443281)

[1.3 The Problem 3](#_Toc68443282)

[2. Selection of Image Scanning Method 3](#_Toc68443283)

[2.1 Raster Scan 3](#_Toc68443284)

[2.1.1 Disadvantages of Raster Scan 4](#_Toc68443285)

[2.2 Random/Vector Scan 4](#_Toc68443286)

[2.3 Decision 4](#_Toc68443287)

[3. Process 4](#_Toc68443288)

[3.1 Software Process 4](#_Toc68443289)

[3.1.1 Path to Points 4](#_Toc68443290)

[3.1.2 Points to Signal 4](#_Toc68443291)

[3.2 Hardware Process 5](#_Toc68443292)

[3.2.1 Engraving Mechanism 5](#_Toc68443293)

[3.2.2 Mirror Control 5](#_Toc68443294)

[4. Components and Tools 5](#_Toc68443295)

[4.1 Hardware Components 5](#_Toc68443296)

[4.2 Software Development Tools 6](#_Toc68443297)

[5. The Software Development 6](#_Toc68443298)

[5.1 Sampling Points using Pulse Code Modulation (PCM) 6](#_Toc68443299)

[5.1.1 Sampling Rate 6](#_Toc68443300)

[5.1.2 Bit Depth 6](#_Toc68443301)

[5.1.3 Quantization 6](#_Toc68443302)

[5.2 Software Development Steps 7](#_Toc68443303)

[6. Discussion 8](#_Toc68443304)

[6.1 Output Quality 8](#_Toc68443305)

[6.1.1 Quality Defining Factors 8](#_Toc68443306)

[6.2 Limitation and Future Improvements 9](#_Toc68443307)

[7. Software Documentation 9](#_Toc68443308)

[7.1 Compatibility and Platforms 9](#_Toc68443309)

[7.2 Development, Debug and Prerequisites 9](#_Toc68443310)

[7.3 User Manual 10](#_Toc68443311)

[7.3.1 SVG to Points 10](#_Toc68443312)

[7.3.2 Points to WAV 11](#_Toc68443313)

[7.3.3 Playback 11](#_Toc68443314)

[8. Bibliography and Acknowledgement 12](#_Toc68443315)

# 1. Project Overview

## 1.1 Objective

The task for this seminar is to design/adopt a hardware and software system for laser engraving. The software takes an image as an input and translates the image data to a format that the hardware can understand. The hardware manipulates the laser to engrave the input image on a surface.

## 1.2 Background

Any greyscale image can be imagined as a set of black points on a piece of white paper. A mathematical point has zero dimension. A mathematical straight line is a set of points whose coordinates satisfy a linear equation. However, a “practical point” in real life can be rather deemed as a circle with a very small radius. Hence, a practical straight line consists of a finite number of practical points. A real-life curve, circle, or polygon can be imagined as a set of many infinitesimally connected straight lines, or moving points.

## 1.3 The Problem

A scenario can be imagined, where A and B are two apparently identical looking curves in a fixed size canvas, but A consists of 10x more straight lines than B. If the canvas is zoomed enough, at some point, the straight lines that form B will be visible to the eye, while the curve A will still look smooth since it has more and smaller straight lines congested inside it. This can be termed as the resolution of the curve. In laser engraving, the resolution, or precision of the engraving depends on many factors. These factors can be determined and influenced by changing various hardware and software parameters of the system that will be discussed later.

# 2. Selection of Image Scanning Method

There are two prominent methods that are used to scan an image: Raster scan and Random/vector scan.

## 2.1 Raster Scan

Raster scan uses pixel-based image scanning. The image is interpreted as a grid of pixels and scanned line by line. The scanning must start top left corner of the image. A horizontal mover sweeps horizontally left-to-right at a steady rate and on reaching the rightmost part of that line, it rapidly moves back to the left. A vertical mover does the same but from top to bottom, at a slower rate. Raster scan is periodic.

### 2.1.1 Disadvantages of Raster Scan

* The engraved resolution depends on the input image resolution. Also, the resolution is dependent on the scanning rate because the scanners may not be fast enough to capture all the pixels of the same line.
* More time consuming since the laser has to be moved for every pixel of the image.
* As an image covers a very small portion of the canvas, the laser is mostly turned off during raster scanning. This leads to huge mechanical and electrical inefficiency.

## 2.2 Random/Vector Scan

Vector scanning looks at an image as a sequence of straight-line segments. A line is just a moving point. Each point is defined by its x & y coordinates on the canvas. The laser can be turned on, placed anywhere on the canvas and moved from one point to the next.

## 2.3 Decision

Vector images are resolution-independent whereas enlarging a raster image lowers the resolution and precision. Vector scan is restricted to polygon drawings, however, that is not a problem for laser engraving purpose. Considering the pros and cons of both methods, vector scanning was chosen for this project.

# 3. Process

## 3.1 Software Process

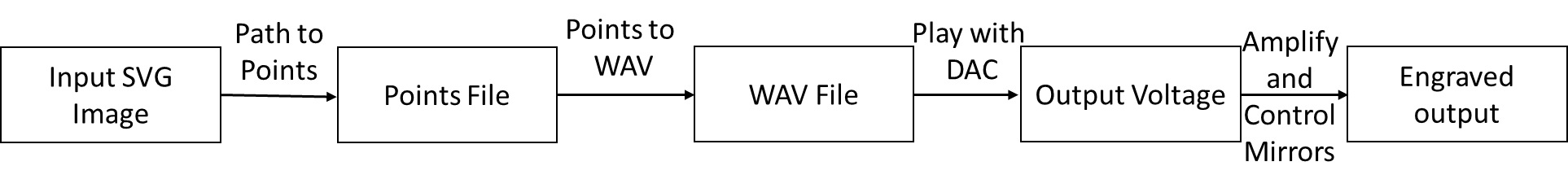
An input image of a Scalable Vector Graphics (SVG) format is necessary since SVG contains points and paths to represent an image.

### 3.1.1 Path to Points

A software was designed and partly adopted to convert an SVG file into data that will be used to control the laser engraver hardware to implement the engraving. The software converts paths i.e., lines of an SVG image into a set of points (x and y coordinates).

### 3.1.2 Points to Signal

The software then samples the points into a digital signal in time domain, where the amplitude of the signal corresponds to the values of the x and y coordinates. The signal is encoded in two channel (left and right channels corresponds to x and y values) Waveform Audio File Format (WAV). This wav audio signal is used to drive the laser hardware to engrave the input SVG image.

 Fig 1: Laser Engraving Process

## 3.2 Hardware Process

### 3.2.1 Engraving Mechanism

To carve an image on a surface, a laser has to be moved to draw the points. Rather than attaching the laser to a moving body, two closely spaced polygonal mirrors mounted on orthogonal axes are used to reflect the laser pointer, where each mirror is driven by a galvanometer. Let x and y be the coordinates of a point on the input image. To draw this point, one mirror has to move x amount along the x axis and the other mirror has to move y amount along the y axis. When the laser is turned on, it gets reflected respectively by the first and second mirrors and eventually falls on the surface and curves the input point. The degree of rotation of the mirrors has to be scaled according to the input image canvas width and height. To carve an entire image, the mirrors move continuously to deflect the laser and draws all the points of the image.

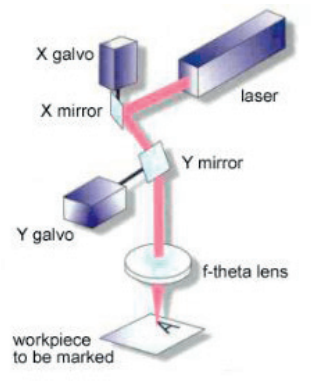


Fig 2: Galvo mirrors to deflect laser [1]

### 3.2.2 Mirror Control

The audio signal that was generated using the software is played via a Digital to Analog Converter (DAC). The output voltage of the DAC is amplified and used to control the galvo mirrors.

# 4. Components and Tools

## 4.1 Hardware Components

* Laser
* Mirror galvanometer system
* Raspberry Pi as Single Board Computer
* HiFiBerry as DAC
* Power supply
* Laser controller
* Cables and connectors
* Oscilloscope for testing the signal

## 4.2 Software Development Tools

* PathToPoints library (javascript)
* C++ for converting points into WAV file
* Shell scripting for test and batch execution
* Visual Studio Code IDE

# 5. The Software Development

## 5.1 Sampling Points using Pulse Code Modulation (PCM)

Digital audio is generated in PCM format. The points that consist an ideal SVG path should be continuous/analog. However, reality is not infinite, thus a path or line can be interpreted as many discrete points. In a PCM stream, the amplitude of the so-called analog signal (in this case, point coordinates) is sampled regularly at uniform intervals e.g., discrete time in x- axis and discrete sample values in y-axis. Each sample is quantized to the nearest possible value. PCM has two basic properties: sampling rate and bit depth.

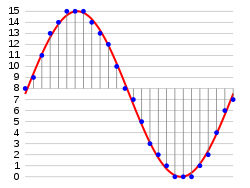


Fig 3: Example of a sine wave sampled with 4 bits (16 levels)

### 5.1.1 Sampling Rate

It means the number of samples per second. While creating WAV file, a sampling rate must be set as a property of the file. For this project, audio files with different sampling rates were created to observe how the DAC handles different sampling rates.

### 5.1.2 Bit Depth

This refers to the choice of number of discrete levels of output values. For this project, 16-bit signed integer has been used. Hence, 216 = 65536 distinct point coordinates can be sampled correctly with this system. It can accommodate cartesian coordinates starting from -215+1 = -32767 up to 215-1=32767.

### 5.1.3 Quantization

The input points are theoretically continuous, so practically they can contain floating point values up to many places after decimal. Quantization process is used to convert these floating point values into nearest integer values.

## 5.2 Software Development Steps

This section discusses the algorithmic steps for generating the audio signals. The entire user guide of the software is discussed in the Software Manual section.

1. SVG image is converted to points (x and y coordinates) using *PathToPoints* library and saved in a text file. Width, height of canvas and an end of file indicator is added manually to the text file.
2. It is highly recommended to add dimensions as *height|width* in the first line of the point text file manually if the dimension is explicitly written in the SVG file. Also, an end of file indicator # should be added in the last line of the text file containing the points.
3. The execution of the code starts from a shell script. It takes the text of the previous step as an argument. If no argument is given while execution, it checks all the text files in the project directory (batch processing) and produces signals only for valid text files (containing proper dimensions, valid point coordinates and end of file marker) and logs the execution process in a log file. If the text file does not contain canvas height and width information (because the SVG file does not contain this information), then it calculates height and width of the canvas by looking at the maximum x and y coordinate values. This process assumes that all the image starts at (0,0) but ends at the maximum values of the coordinates. This is not a perfect solution, but dimension calculation is necessary as a fallback since SVG files often do not have a fixed height and width properties. After validating the file/files, the script executes a C++ program that converts the point coordinates into a WAV signal.
4. The C++ program reads the text file containing the points and loads all points in a memory (vector of point objects)
5. The points from the text file contains non-negative coordinates since the graphical coordinate starts at (0,0) from top left and positively increases along both axes. Each coordinate value is normalized e.g., feature scaled between 0 and 1
6. After normalization, each point is scaled between [-5, 5] so that the entire range of 16-bit signed integer (positive and negative) can be used.
7. Each coordinated value is quantized i.e., multiplied by a constant 60000 (range [-30000, 30000]). This constant must be smaller than ~65500 if for a bit depth of 16. The quantized values are samples for the audio signal.
8. A dynamic array of samples is used to hold the integer values of the signal. A default time of 10 seconds and a default sampling rate of 48000 Hz is used as the signal parameter. However, all parameters of the signal are changeable input parameters, so that the software can be tested flexibly. The array has a large maximum size, so, in most cases it should be able to contain all the points i.e., samples values. If there are not enough samples to fill the array, linear interpolation technique is used to calculate the sample points between two adjacent points.
9. Two buffers contain all the samples: one is used to store all x-coordinate samples and the other is used to store the y-coordinate samples.
10. A new WAV file is created with proper headers. The two buffers containing the signals are written to the WAV file: one buffer is written to the left channel and the other signal is written to the right channel. These channels will be accessed by the left and right channels of the DAC.
11. The WAV files are generated with different sampling rates for the same image. These WAV files are placed in the storage of a Raspberry PI (RPI). A HifiBerry DAC is attached and configured with the RPI. A simple C code, or any software such as “aplay” is used to play the WAV file via the DAC. The DAC smooths out the digital signal by converting it to analogue format.
12. N.B. the generated signal draws the image two times, from starting point to end point and then end to starting point. It is not necessary for this project. However, this periodic drawing allows the code to generate a laser show just by providing a large input frequency. Moreover, the first 100 points of the signal contains a threshold samples for triggering purpose in oscilloscope.

# 6. Discussion

## 6.1 Output Quality

The quality of the engraving depends on many factors. While engraving a curve, the laser is actually drawing many small straight lines. Hence, it is possible to see jumps from one point to the next in the output. This breaks the smoothness of the drawing. The DAC smooths out the jumps, but it might also deviate the output from the original image if the jumps between points are too big.

### 6.1.1 Quality Defining Factors

* **Number of Points**: As the input image is converted into a set of points, the number of points is proportional to the quality of the output signal. More points mean that they are congested more closely to each other, hence it reduces the jump size between two adjacent points.
* **Quantization**: Bit depth of the digital signal is also proportional to the output quality. If an image is converted to too many points, the points must have very close coordinates. During quantization, the precision of the coordinates is lost. A higher bit depth can perform better quantization. However, increasing number of points and bit depth consume more resource.
* **Galvo Mirrors**: The mechanism of galvo mirrors is also a defining factor for output quality. A very high precision of the rotation is necessary to draw closely related points.
* **Playback Parameters**: Bit rate (bits per second = sampling frequency \* bit depth \* No. of channels) of the signal is important during playback of the signal. If the signal’s sampling frequency is too low than the DAC’s sampling frequency, then the DAC will add interpolation to smooth out the signal, which might deviate the output from the actual input image e.g., the jumps between points may be visible. However, if the signal has a higher sampling rate than the DAC, then some samples are lost during playback, so the output shrinks.
* **Distance**: The distance of the surface on which the carve happens also affects the quality of the output. The farther the surface is from the laser, the more visible the jumps will be between adjacent points.

## 6.2 Limitation and Future Improvements

The software is designed to create signal for a single closed shaped polygon. If there are several disconnected polygons in the image, the *PathToPoints* shows coordinates for each closed polygon separately. Each points file can then be processed individually to make distinct WAV files. In future, a different plan for interpreting the input image can be devised to enable the handling of all the shapes in a single audio file. In such case, it will be necessary to turn on and off the laser between drawing disconnected closed shapes.

# 7. Software Documentation

This is a guide for development, debugging and using the audio signal generator software.

## 7.1 Compatibility and Platforms

* C++17 was used to develop the software
* Visual Studio Code was used as IDE
* Development and execution tested for macOS (11), Ubuntu LTS (20.04) and Windows 10
* Compilations tested: gcc 9.3.0 on Ubuntu, Clang on MacOS, and gcc 10.2.0 (MSYS2) on Windows
* Tested on Intel Core i7 and i5 Processors
* Shell script execution tested on Windows PowerShell, Mac and Ubuntu Terminal and git bash
* All tests were carried on 64-bit OS and 64-bit processor

## 7.2 Development, Debug and Prerequisites

* Install git and download the project from github using git bash with the command *git clone https://github.com/sharifkaiser/laser\_seminar.git*
* **Windows**:
  + Download Visual Studio Code and install C/C++ extension for VSCode. Create a new directory *.vscode* under the project directory (laser\_seminar) and copy all files from *windows.vscode* to *.vscode* directory.
  + Install MSYS2 and all the toolchains by following all the steps from the official page <https://www.msys2.org/>
  + Using MSYS2 MSYS, install gcc compiler and gdb debugger with these two commands: *pacman -S mingw-w64-x86\_64-gcc* , *pacman -S mingw-w64-x86\_64-gdb*
  + Add the path *C:\msys64\mingw64\bin* to windows Environment variable 🡪 System Variable
  + Install Bash Calculator (bc) using MSYS2 MSYS command *pacman -S bc*, and then copy *bc.exe* from mysys64 installation path e.g., *C:\msys64\usr\bin\bc.exe* to git installation path e.g., *C:\Program Files\Git\usr\bin\* to be able to run shell script error free from git bash.
  + Now, build and debugging from Visual Studio Code should work
  + Execute the shell script with command *./test.sh* from either git bash or Windows Power Shell
* **Ubuntu**:
  + Download Visual Studio Code and install C/C++ extension for VSCode. Create a new directory *.vscode* under the project directory (laser\_seminar) and copy all files from *ubuntu.vscode* to *.vscode* directory.
  + Install Bash Calculator (bc) from terminal using command *sudo apt-get install bc*
  + Now, build and debugging from Visual Studio Code should work
  + Execute the shell script with command *./test.sh* from ubuntu terminal
* **MacOS**:
  + Download Visual Studio Code and install C/C++ extension for VSCode. Create a new directory *.vscode* under the project directory (laser\_seminar) and copy all files from *mac.vscode* to *.vscode* directory.
  + Install Bash Calculator (bc) from macOS terminal
  + Now, build and debugging from Visual Studio Code should work
  + Execute the shell script with command *./test.sh* from macOS terminal

## 7.3 User Manual

### 7.3.1 SVG to Points

Under *laser\_seminar/PathToPoints-master* directory, open *index.html*. Upload the input SVG file and set *Point every x length* text input as *1* since this generates the maximum number of points. If the SVG contains many polygons, points for every polygon will be generated separately. Please notice the matched colour and copy all the points for the desired polygon. Create a text file under the project directory and paste all the copied points into the text file. Add a ***#*** at the last line to denote the end of file. The last line should not contain any point. If the input SVG has explicit height and width properties, append that information to the first line of the text file as *height|width*. If the SVG file does not have these properties, or it is not clear from the SVG file whether the height and width actually represent the canvas height and width, it is better not to add the dimensions to the text file. The software automatically adds a probable dimension during the execution process. A typical text file that containing three coordinates might look like the following:

*600|400*

*47.550,89.6*

*58.4589,93.012*

*40.1,66.9878*

*#*

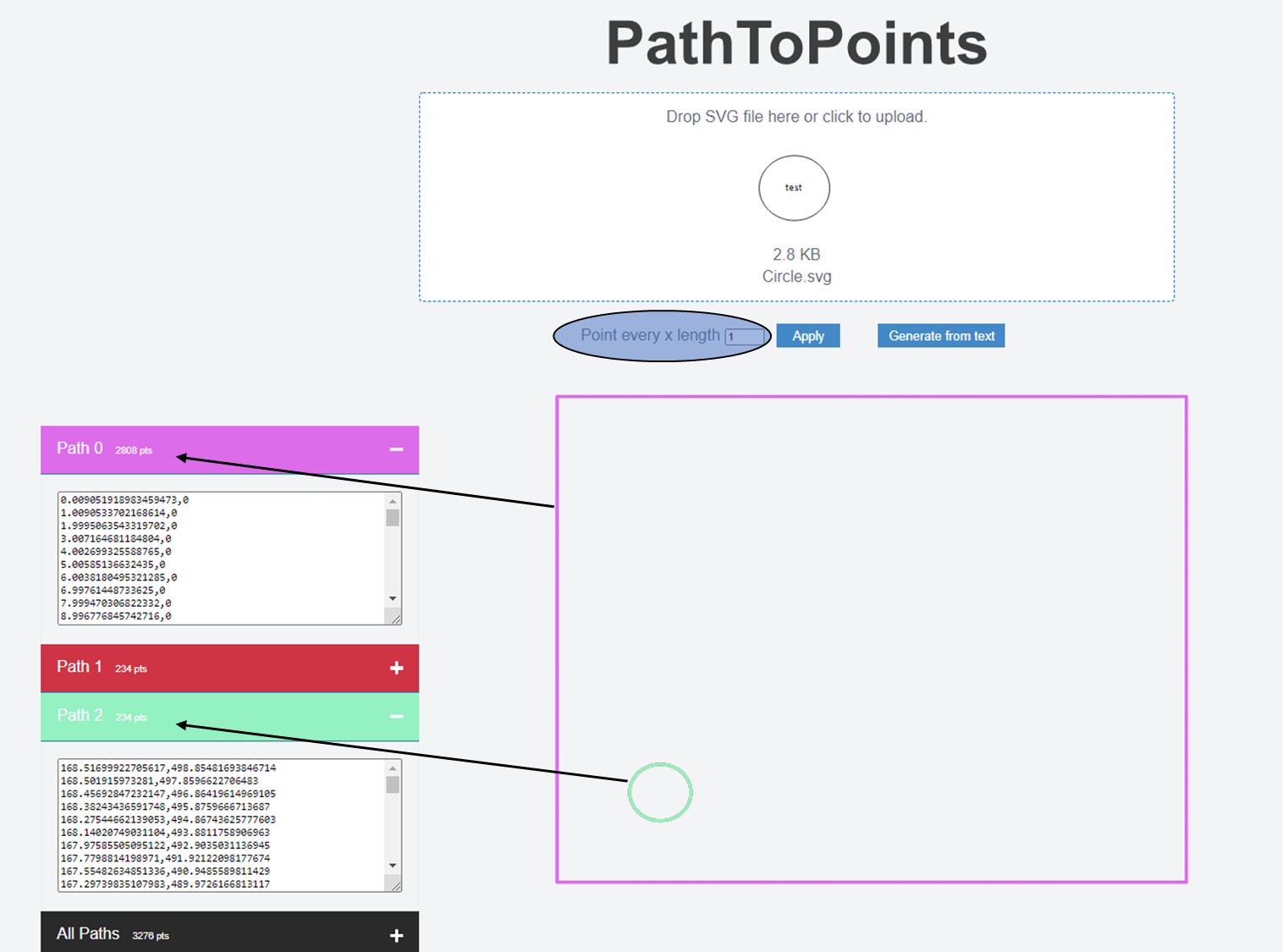


Fig 4: Converting SVG Paths to point coordinates

### 7.3.2 Points to WAV

Open terminal (MacOS or Ubuntu), or git bash/PowerShell on Windows and change directory to the project directory and run the following command to create audio signal:

*./test.sh <point\_file.txt>* # Processes single file

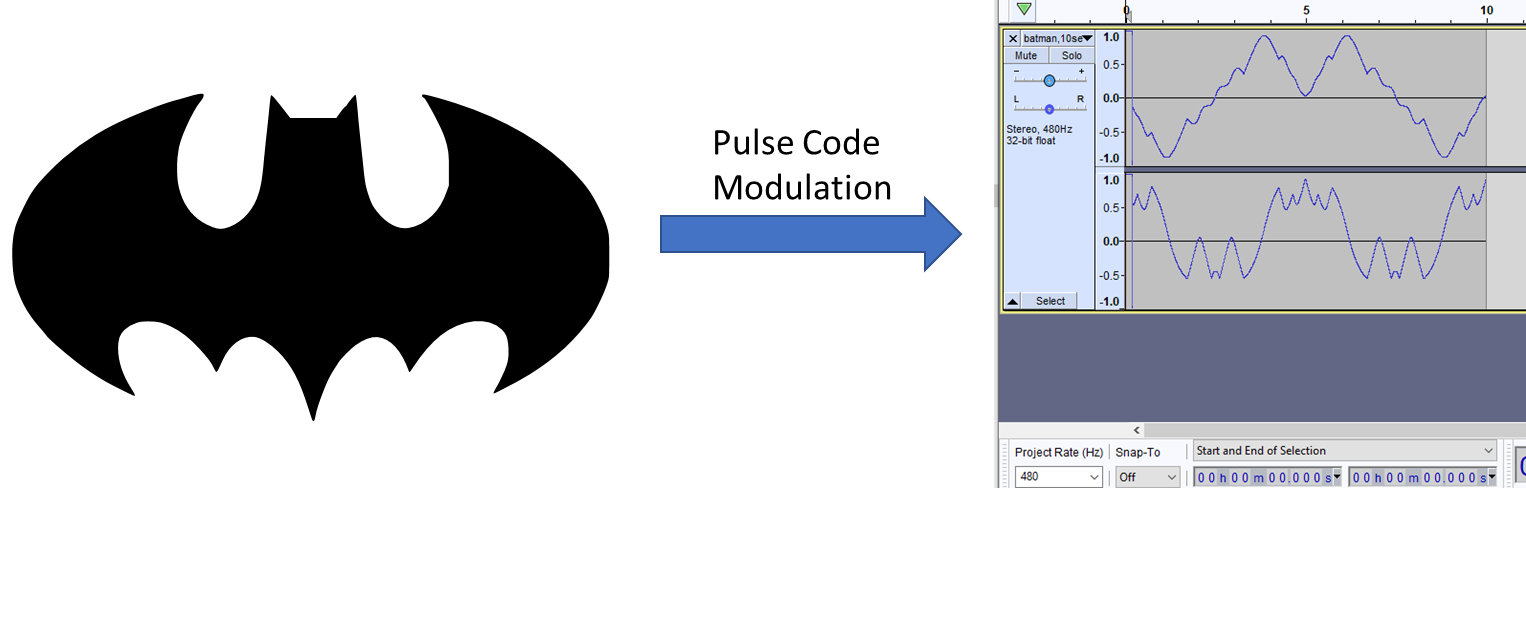
*./test.sh* # Batch processing e.g. all text files in current directory

For each text file, several signals with different sampling rate and bit rate are generated.

### 7.3.3 Playback

Transfer all the generated WAV files to the Raspberry Pi. Connect HiFiBerry with Raspberry pi and configure it from the official webpage <https://www.hifiberry.com/docs/archive/hifiberry-software-configuration/> and Play wave file using *aplay <wav\_filename>*

It is also possible to observe the signal using ***Audacity*** open-source software.

 Fig 5: SVG to WAV

# 8. Bibliography and Acknowledgement

1. Polygon scanner system for ultra short pulsed laser micro-machining applications - R. De Loor

2. Raphael Monnerat for PathToPoints library: <https://github.com/Shinao/PathToPoints>

3. Dr.-Ing. Detlef Streitferdt and Fachgebiet Softwarearchitekturen und Produktlinien for providing guideline and instruments