UNIVERSITÄT
BAYREUTH

Fakultät Mathematik, Physik, Informatik
Institut für Informatik
Lehrstuhl für Serious Games
Prof. Dr. Jörg Müller

Big Master Project

The Haptic Printer

Author : **Anuj Sharma**,
Akshaya Kumar Krishnappa Ramakrishna

Supervisor : **Viktorija Paneva**

September 27, 2021
Version: Final

Abstract

Over the years of evolution of human computer interaction, we have seen various kinds of interaction techniques. The recent trend of contactless haptic technology has been on the rise. In our project we focus on rendering custom shapes on the user palm, using different kinds of rendering techniques. The main objective of this project is to map various kinds of user input like CSV, SVG, drawing from canvas and inputs from leap motion. We consume these inputs and process them in order to render custom shapes. Also, this report analyses and compares the different techniques of rendering with various validation techniques and deduces an overall result from the study and implementation of the proposed application.

Contents

1 Introduction

The haptic technology enables human computer interaction in a new way compared to previous existent interaction techniques. Haptics gives us a new dimension of interaction technique without any physical contact between the user and the device. This contactless haptic technologies give rise to various kind of applications in real life scenarios. One prominent among them is the mid air user interfaces, where the use of hand and finger movements are used to control the interface. Ultrasound haptic technology [ITS08] is one of the example of contactless haptic technology where ultrasound waves are emitted by the device and coincide at a particular point in space to give haptic sensations.

1.1 About Ultrahaptics

The human skin has mechanoreceptors which are sensory cells that convert mechanical forces into nerve excitation. When the sound waves make contact with the skin, it gives a tactile sensation for the user. In order to produce such ultrasound waves we make use of a device manufactured by Ultraleap [Ul] called as Ultrahaptics. The device emits ultrasound from its phased array which propagates as a wave with a frequency higher than the limit of human hearing. Each emitter in the device emits the wave and every wave emitted coincide at a point in the air. This coincidence of the waves at a point is called a focal point and this focal point when directed towards a user palm creates a pressure for the user and the user can perceive it as a tactile sensation. The coincidence of waves at a focal point, strength and intensity of each transducer is handled by the Ultrahaptics device itself by using various solver algorithms.

In order to render custom shapes we need to explore various techniques of rendering and experiment with different parameters of the device. The different parameters available in the device are intensity, frequency, multiple focal points and the different techniques of rendering are explained in the next section of literature survey.

1.2 Project objective

Using the concept of mid air interfaces, we can explore and build many applications in real world. In the recent times due to the COVID-19 pandemic, one prominent use case would be to build contactless interfaces in the public space. In order to achieve this we need to render custom shapes to the user in order to make user aware of different kinds of sensation. In this project, our main focus is to render custom shapes on user's palm using different kinds of inputs. Here, different kinds of inputs means different various formats like CSV (comma-separated values), SVG (Scalable Vector Graphics), user drawing and Leap Motion. We process all these kind of inputs and render the dynamic shapes using the Ultrahaptics device.

2 Literature Survey

Once our project objective was clear we had to find out existing rendering techniques available for Ultrahaptics device. Also, we had to look into Ultrahatics Software Development Kit (SDK) and the available Application Program Interface (API) so that the application can be developed. After referring to Ultrahaptics SDK we came to know that there are two techniques available to develop our application, namely Amplitude Modulation (AM) and Time Point Streaming(TPS). To further understand these topics we had to look into the concepts and working of these techniques. So we did a literature survey and in the further subsection we have listed our key findings and short explanation of both the techniques.

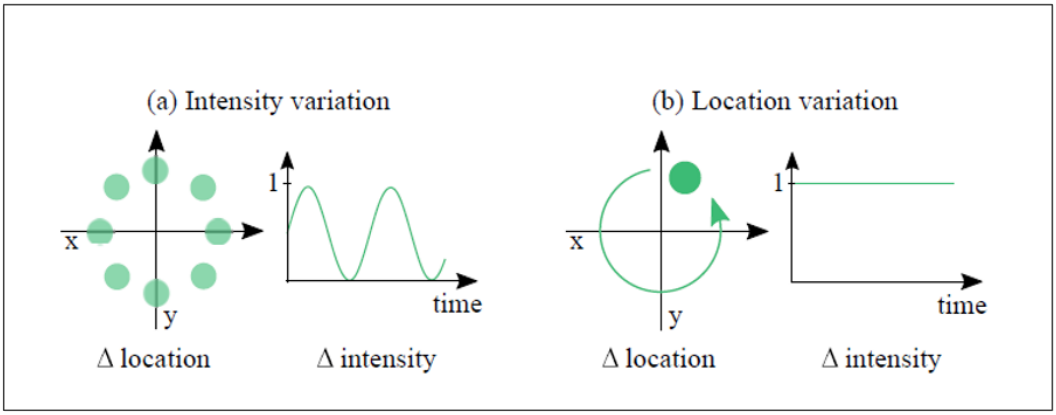

Fig. 2.1: (a) shows working of Amplitude Modulation technique (b) shows working of Spatiotemporal Modulation technique [Fri+18]

2.1 Amplitude Modulation

In amplitude modulation technique, the ultrasound waves emitted by the phased arrays and are modulated by switching them off and then on again fast enough to stimulate the vibration-sensitive touch receptors (mechanoreceptors) in the skin of the hands, giving tactile sensation. But these switching on and off of waves are not so easily perceived by the user as the frequencies are in the range of 40Hz to 400Hz. In order to smoothen out this effect the ultrasound is modulated with a sinusoidal wave.[Am]

As shown in Fig 2.1 (a) the control points have varying intensity emitted by the device which creates pressure points on user's palm. Using this technique, we can

render shapes by updating the coordinates of the shape rapidly. Also, we can use one more technique here by emitting multiple control points at the same time to complete the shape outline.

2.2 Spatiotemporal Modulation

Spatiotemporal Modulation also called as time point streaming is a technique where we rapidly move the control point emitted by the device along the shape outline to create tactile sensation. Here the control point can be fixed to a same intensity level or we can use any sine or cosine wave as intensity values for varying intensities.[Tps]

As shown in Fig 2.1 (b) the control point moves rapidly along the circle shape outline with constant intensity.

Once we discovered the various techniques available in Ultrahaptics SDK we decided to go ahead with these two techniques. There are several tweaks available to these two techniques which can also be used for rendering. However, due to the limitation of available Ultrahaptics SDK API we chose to stick to these two techniques.

3 Features

> *A good end product is not just a collection of features, Its how it all works together.*
>
> **— Tim Cook**
> -CEO Apple Inc.

As mentioned previously the main objective of the application is to consume multiple types of inputs from user and render it to the Ultrahaptics device. The features are built keeping those multiple inputs in mind. A display of multiple inputs can be seen in the image below. The features that are supported in the application are

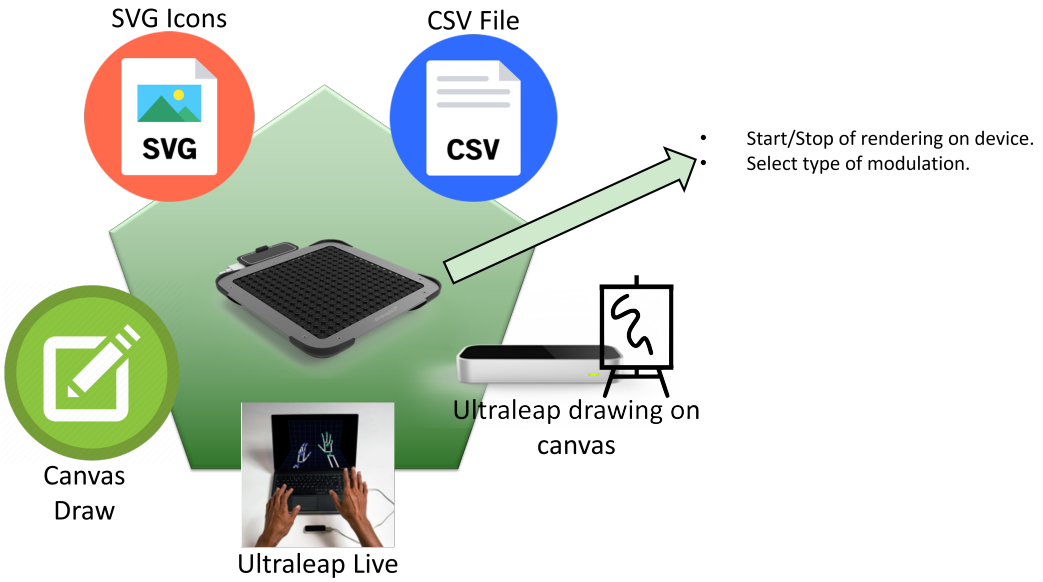

Fig. 3.1: Figure: Features of the application

categorized in three part:

a) Output: Ultrahaptics rendering.
b) Inputs: multiple types of inputs supported.
c) Controls: how user can control rendering.

3.1 Output

Since this application revolves around one major output, it is important to mention the output in the beginning. The single output of the application is the custom dynamic shape to be rendered/emitted by the ultrahaptics device. Since, the device has limited size each input coordinate/s should be scaled accordingly.

3.2 Inputs

Multiple types of inputs are accepted in the application and tabs are created for different inputs, these tabs can be seen in fig 3.2:

Fig. 3.2: Tabs in WebApp

CSV File

A CSV file which has the coordinates of points in an x,y plane in column family. It can be uploaded in File tab. It is assumed that the coordinates are mentioned in meters. But since the size of ultrahaptics is 16mm $\times$ 16mm the values in CSV are expected to be in that range. For eg the coordinate (4,5) mm should be 0.004, 0.005 in CSV file. An example of the csv input plotted on an xy plane can be seen in figure 3.3 only for the visual purpose.

In other forms of inputs to the application, the coordinates are scaled and centered

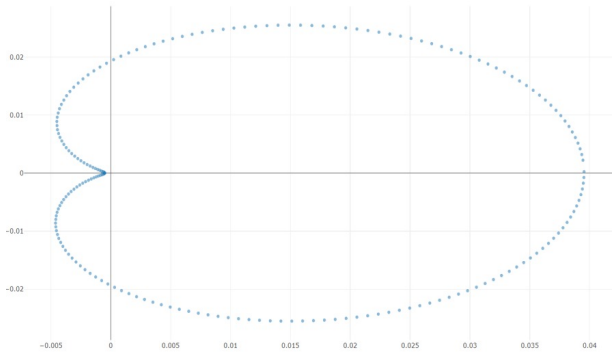

Fig. 3.3: Example csv Plotted on xy plane

to size of the Ultrahaptics device by the application. However in csv input, it is assumed that the coordinates are scaled and centered. As soon as the user clicks on

the render button the coordinates are transferred to the Ultrahaptics device and it starts rendering.

SVG File

An SVG is also accepted as input format in the same tab as csv. As soon as the user uploads the SVG its thumbnail is shown on the same page to confirm the visibility. SVG coordinates are then communicated to the ultrahaptics device for that shape to be emitted after user clicks on render button.

Canvas Draw

This feature can be accessed under Canvas tab. A HTML canvas is provided in which user can draw basic shapes and patterns. The coordinates from the canvas is then communicated to the backend and rendered on the ultrahaptics device.

Ultraleap Live

Ultraleap[ultraleap.com] is an advanced hand tracking hardware sensory device that accepts hand and finger gestures as input, similar to a mouse, but without the need for physical contact. It is is a tiny USB peripheral device that is meant to be put on a physical desktop with its face forward. It's also compatible with virtual reality headsets. The gadget observes a roughly hemispheric region to a distance of about 1 meter using two monochromatic infrared cameras and three infrared LEDs[Wei+13]. The LEDs emit patternless infrared light, and the cameras capture almost 200 frames per second of reflected data[Lea]. This is then transferred to the connected computer via USB connection, where it is processed by the company's proprietary hidden code, synthesizing 3D position data by comparing the 2D frames recorded by the sensors.

Fig. 3.4: Leap motion controller

We consume the data provided by the leap controller and mimic the motion, movement and direction of moving index finger on the ultrahaptics board. Another requirement of the feature is also an ultraleap device connected and setup on the system where the webpage is accessed. This feature is accessible in the tab leap live.

Ultraleap Canvas Draw

Similar to the ultraleap live user can draw shapes on a canvas provided in WebApp using the leap motion controller. As soon as the user clicks on render button the shape is rendered on ultrahaptics board.

Note: To provide input through leap motion controller device, click on the Start Leap button provided in our WebApp and then do the click/tap gesture by your index finger above Ultraleap motion controller device. After that you will get a banner notification that the system is tracking your finger.
Do the similar gesture to stop the tracking.

3.3 Controls

Some basic controls are also provided in the applications where user can select and control other factors, such as:

Stop button:

On top right of our WebApp a stop button is provided, the rendering on the ultrahaptics device can be stopped anytime using this button.

Selecting type of rendering:

On top left of the webapp there is a dropdown select. Here user can switch between Amplitude Modulation or Time Point streaming for rendering. Once user has selected AM, and now wants to observe TPS, rendering has to be stopped by the stop button before switching.

4 Architecture

> *Any application that can be written in JavaScript, will eventually be written in Javascript.*
>
> — **Jeff Atwood**
> Code Horror (blog)

This section describes the usage and mapping of hardware and software of this application. This section will clear out some questions about the application like, how the application is consuming the inputs? How the application is transforming input to the desired output? How are the middle layers processing the data? Including some other questions about communication of different components. The aim of this section is to explain in detail the architecture of the application and flow of features. This will also explain the requirements to execute the application on local system.

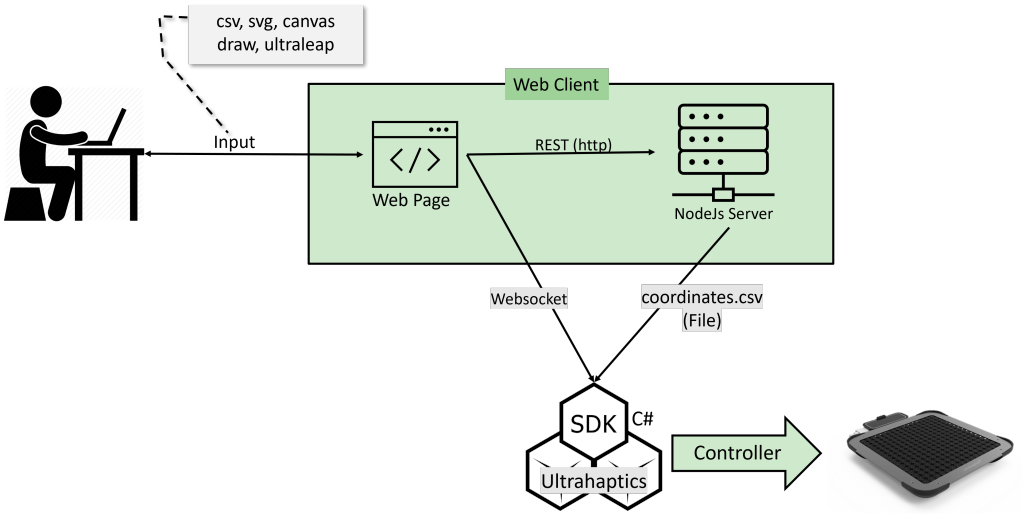

Fig. 4.1: The Haptic Printer (Application architecture)

As seen in the architectural diagram Fig 4.1 there are multiple components of the application, from user input to ultrahaptics device to the communication through websocket and REST. The components are mentioned in detail below:

4.1 Frontend

The front end of the application is the web. The web application is built in vanilla-JS served over a nodeJs Server. However, for compiling js code for browser babeljs[Bab] is used.

The WebApp and Inputs

The webpage is responsible for taking multiple inputs from the user and start and stop Ultrahaptics device. The inputs are in form of the canvas draw, SVG icons, CSV file with (x, y) coordinates, Ultraleap inputs. The web page is responsible to scale and center the coordinates (according to x/y plane) taken from multiple input types to the Ultrahaptics dimensions and units.
In case of SVG some more processing is required. Since, there is no direct feature provided by the browser to extract the coordinates from an SVG. The code written for this is a custom implementation inspired from a spotify tool[Spo] which is under Apache license 2.0 . After extracting the coordinates from the given input the coordinates are scaled according to the output device and sent to the backend server in NodeJs, the processing in the nodeJs is explained in the section backend.
After, the successful response from the Node Server the front end is also responsible to communicate to the Ultrahaptics SDK implementation to start and stop the Ultrahaptics rendering.

4.2 Backend

The Backend is further divided into two parts. The First, is the Node Js Server. Second is, the system implementation in C-sharp, which is responsible to control and render shapes in Ultrahaptics device.

NodeJs Server

The functionality of nodeJS server is minimal. It hosts the WebApp at localhost:3000 in its root and also consumes the scaled coordinates from the WebApp. After getting the coordinates it creates a CSV file and writes all the coordinates in a csv as x,y as two columns. Post this functionality it responds to the webapp with a success message. This CSV file is later consumed by the SDK API implementation to render it on Ultrahaptics device.

The SDK extension: Ultrahaptics API's

Ultrahaptics SDK[Ultb] has provided some of the classes to manipulate the control points, intensity and how the emitter will respond to the control points and positions provided to it. We built a system using these API's to continuously render custom points and shapes provided. All the previous implementations and tests we found were for a shapes which were pre defined in the code and it was a challenge to dynamically change the emitter points at a given time.
This part of the system implements two types of rendering Amplitude Modulation(AM) and Time Point Streaming(TPS). Both the implementations consumes dynamic inputs and detail explanation of how it is implemented can bee seen below in code documentation in Files AM.cs and TPS.cs. The choice of rendering type between AM and TPS can be selected from the WebApp.
The system also implements a websocket for communication which will be discussed further.

4.3 Communication

Websocket

The websocket communication happens between the web app and Csharp Ultrahaptics system to consume API's. Where Csharp implements the websocket server $UltrahapticsOrchestrator.cs$ and webapp implements the websocket client $cswebsocket.js$. The webapp sends a message to backend system to start or stop the Ultrahaptics device rendering, using websocket. It also sends what type of rendering to use(AM or TPS). the websocket communication message is in JSON format and an example message can be seen below.

```
{
  "action":"start",
  "type":"AM"
}
```

Fig. 4.2: Json example to start AM rendering

REST

The REST communication in between the webapp and the NodeJs server. After the input processing in the webapp the coordinates are sent to the NodeJs using this REST channel at $localhost : 3000$.

CSV file I/O

After the NodeJs server receives the coordinates, the coordinates are written to a csv file in the column format. Column names (x,y). These coordinates file is picked up by the Ultrahaptics API implementation system, and uses these coordinates to render the mentioned points.

5 Validation

After the development of our web application it was time for validation to check how the custom shapes are being rendered from the Ultrahaptics device. In order to validate our rendered shapes we shortlisted three different techniques which are explained in detail in the following sections.

5.1 Ultraviz tool

The Ultraviz is a tool developed by Ultraleap [Ul][Ulta] to visualize control points that are being rendered by the Ultrahaptics device in a three dimensional space. It helps the developer to understand how the rendering is happening visually. With the help of this tool we tried to analyse the different rendering techniques like AM and TPS with different kind of inputs i.e, CSV, SVG, drawing on canvas, live leap motion and drawing on canvas with the help of leap motion.

The following are the various kinds of inputs we tried to render:

Note: Since the images are screenshots of the Ultraviz visualization tool, we cannot see all the control points being rendered in the image as it is moving at a high speed.

1. Read CSV file feature:

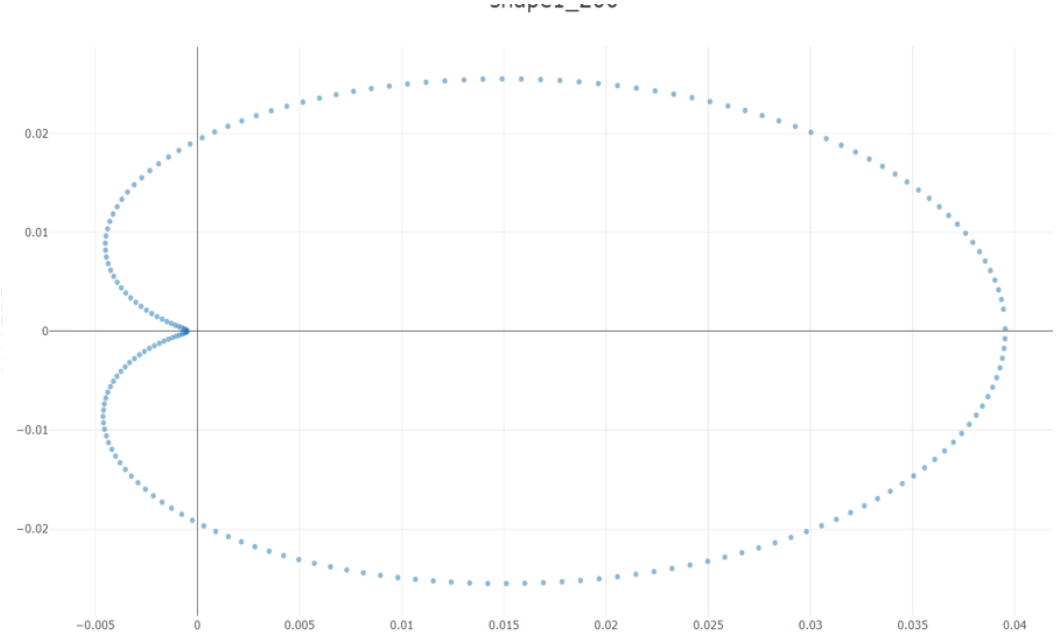

Fig. 5.1: CSV Plot of a custom shape

The x and y coordinate values are read from a CSV file and the plot is as shown in Fig 5.1. This plot will be rendered in the Ultrahaptics device. Fig 5.2 (a) represents the CSV plot being rendered on Ultrahaptics using AM technique and Fig 5.2 (b) represents the same CSV plot in TPS technique.

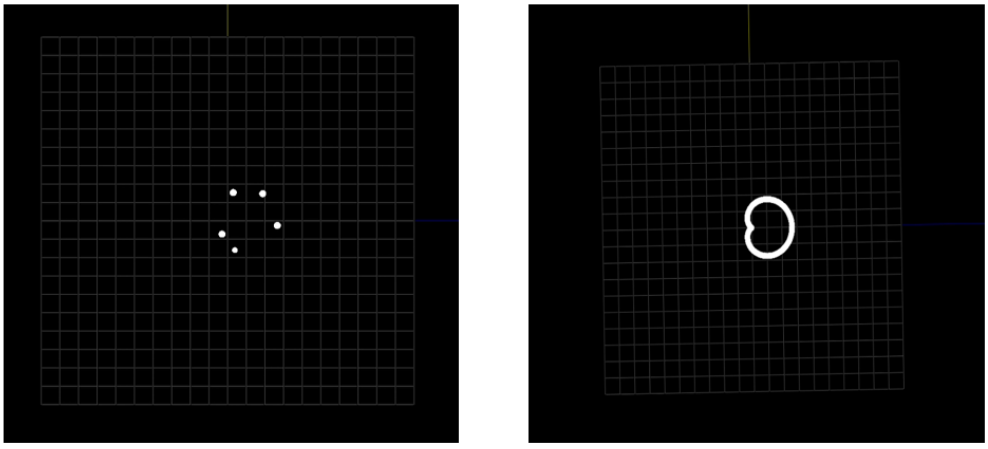

Fig. 5.2: (a) Ultraviz visualization of CSV plot in AM and (b) TPS technique

2. SVG file input feature:

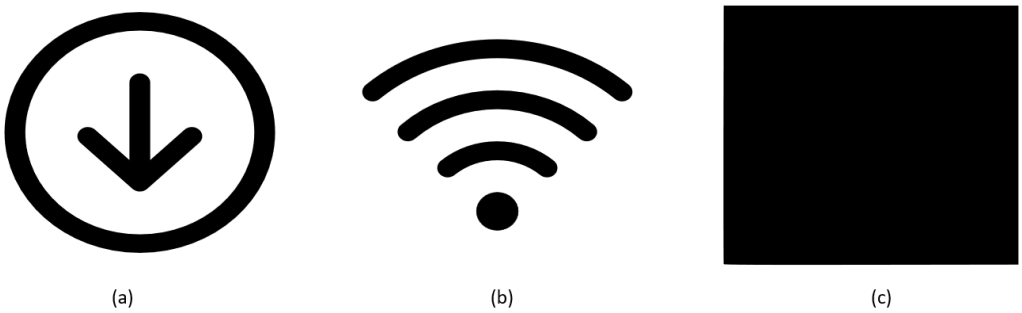

Fig. 5.3: SVG input shapes (a) Arrow Down (b) WiFi (c) Square

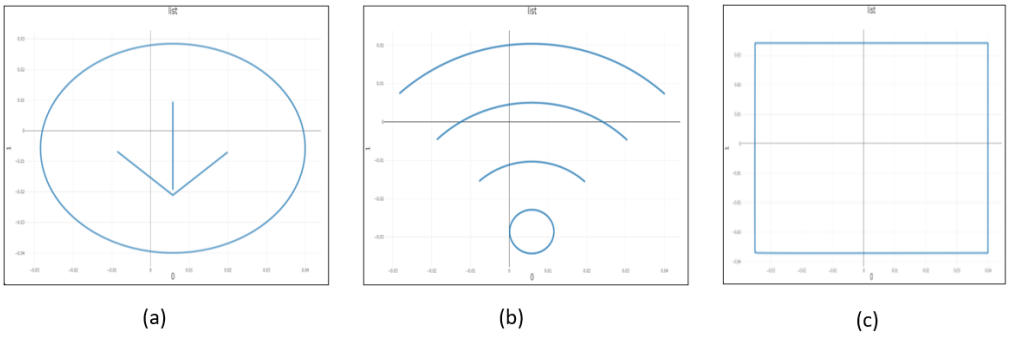

Fig. 5.4: SVG plot of input shapes (a) Arrow Down (b) WiFi (c) Square

Fig 5.3 depicts the various SVG files used to render on Ultrahaptics device. Once the SVG files are uploaded it is converted to x and y coordinate system with specific number of points. Fig 5.4 shows the different xy plot of the SVG files uploaded.

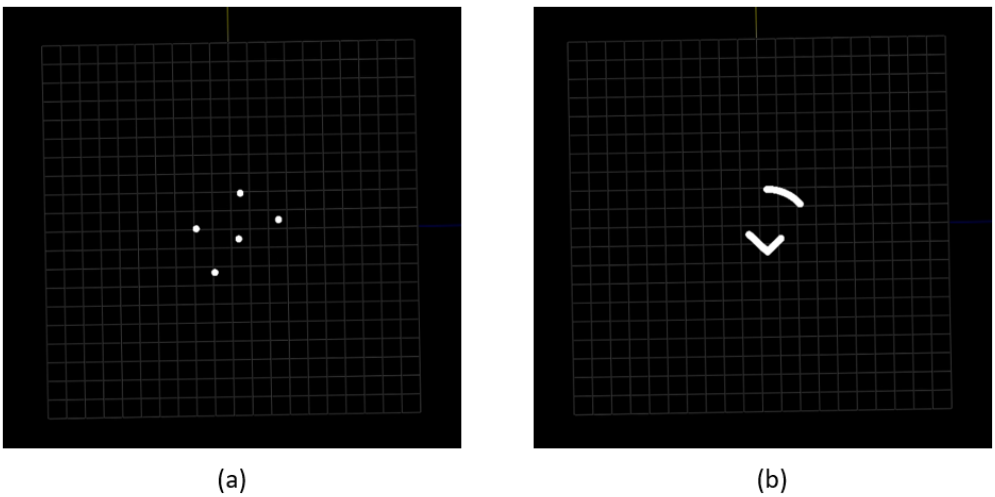

Fig. 5.5: (a) SVG rendering of arrow down in AM (b) SVG rendering of arrow down in TPS

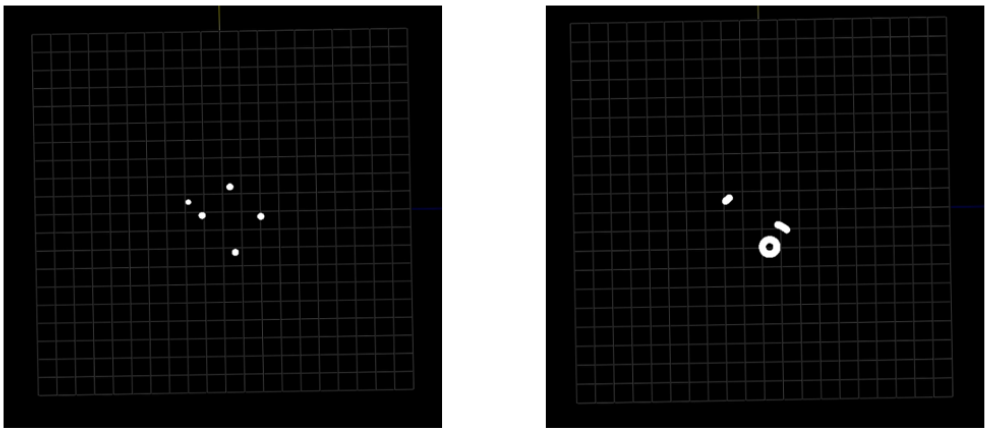

Fig. 5.6: (a) SVG rendering of WiFi in AM (b) SVG rendering of WiFi in TPS

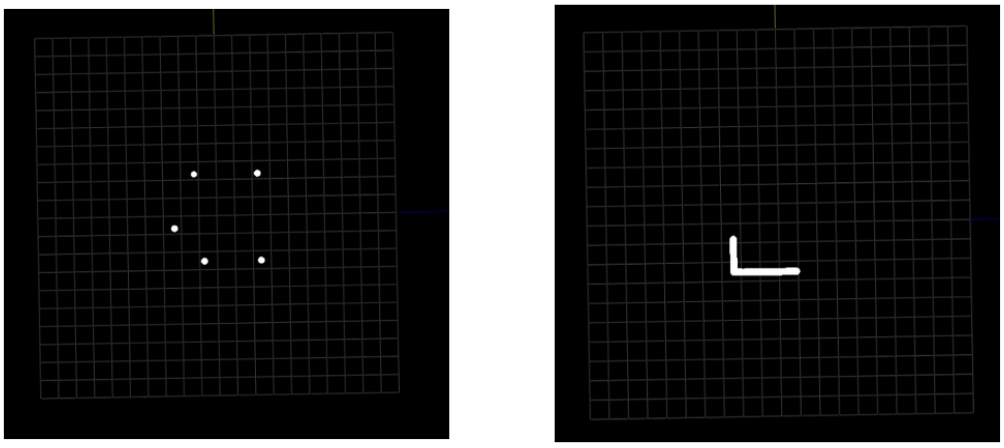

Fig. 5.7: (a) SVG rendering of square in AM (b) SVG rendering of square in TPS

Fig 5.5, Fig 5.6, Fig 5.7 displays an image of Ultraviz tool rendering the SVG inputs arrow down, WiFi and square in AM and TPS technique respectively.

3. Drawing on canvas:

The user can draw the shapes on the given HTML canvas using mouse. Fig 5.8 (a) shows a user drawn circle shape and Fig 5.8 (b) shows an plot of the user shape represented in x and y coordinate which will be eventually rendered on Ultrahatics.

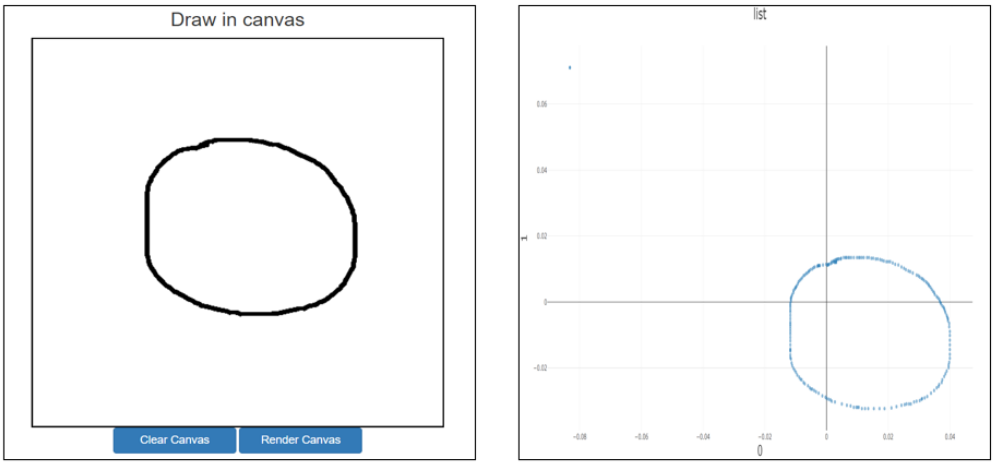

Fig. 5.8: (a) User drawn circle shape on HTML canvas (b) XY plot of the given user shape

Fig 5.9 (a) depicts the rendering of custom user shape in Ultraviz in AM and Fig 5.9 (b) depicts the same in TPS technique.

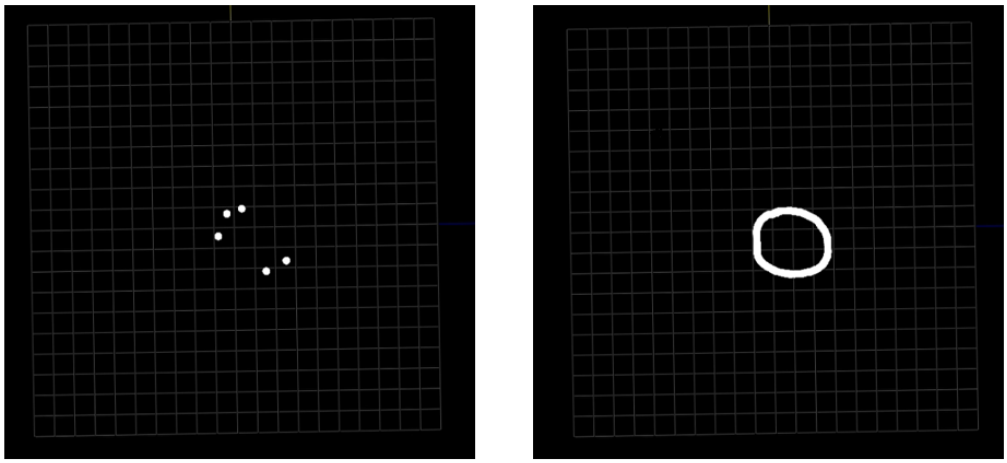

Fig. 5.9: (a) User drawn circle rendering in AM technique (b) User drawn circle rendering in TPS technique

4. Leap Canvas:

Leap canvas is similar to canvas drawing, instead of using mouse as a drawing tool here we use our index finger and with the help of Ultraleap leap motion device we draw on the canvas. Later the canvas drawing is converted to xy plot and then rendered on the Ultrahaptics device.

5. Leap Live:

Leap live uses the Ultraleap leap motion device to track our index finger and the same point is being rendered live on the Ultrahaptics device. The control point moves in the direction of the Ultraleap leap motion hand input.

5.2 Cross validation and observation from developers

In order to further validate our project we conducted a short study between the authors of this project where we chose three different shapes i.e, square, circle and triangle and decided to guess the shape in various different techniques. In the first study we did not tell the observer on the Ultrahaptics which all shapes where there in the study and the observer guessed everything as circle.

Later in the second study, we informed the observer the available shapes and then performed the study. We conducted the study mutually between us and below are the results: Square shape was easily identified by both of us in AM technique and both guessed the square shape as circle in TPS technique. Triangle was guessed correctly in both the techniques. Circle was guessed correctly in TPS technique and one observer reported circle as triangle in AM technique.
Hence, from this short observation it was concluded that basic shapes if known to the subject in advance are recognized in a better manner.

Below in Fig 5.10 you can see all the results summed up in a table.

Observer: Anuj		
	Amplitude Modulation	Time Point Streaming
Shape		
Square	Yes	Answered as circle
Circle	Yes	Yes
Triangle	Yes	Yes

Observer: Akshaya		
	Amplitude Modulation	Time Point Streaming
Shape		
Square	Yes	Answered as circle
Circle	Answered as triangle	Yes
Triangle	Yes	Yes

Fig. 5.10: Results of validation between the authors of the project

5.3 Oil bath

We also came across a technique where the rendering can be projected on an oil bath set-up to visualize how the inputs are being rendered on the user palm [Oil]. This experiment was a guided setup by the ultrahaptics support team. The step by step guide to set up the environment is as follows:

1. The Ultrahaptics device is suspended approximately 15cm above the oil bath using a robotic arm. There is no need to use the Leap Motion so the cradle can be separated.

2. The oil is between 2 and 5mm deep. There is a purpose built perspex tank of 30 centimetres for the experiment. The oil must have the correct consistency and viscous enough to show dispersion, fluid enough to be responsive. We have found that a 50:50 mix of olive oil and pumpkin seed oil give good results.

3. The oil bath must be raised approximately 3 cm above a white surface. We used legos for this purpose and used two A3 sheets of paper as a white surface.

4. Light with a single, small, bright light source from above or to an angle to project the shadow of the distorted surface on to the oil. There is a small, LED light available on an adjustable stalk. Place it between top of tank and perspex holder, with light covering as much of oil as possible. The light must be bright enough to reflect onto a wall and the room must be as dark as possible to achieve the best affect.

Fig 5.11 and Fig 5.12 show the experiment setup of oil bath.

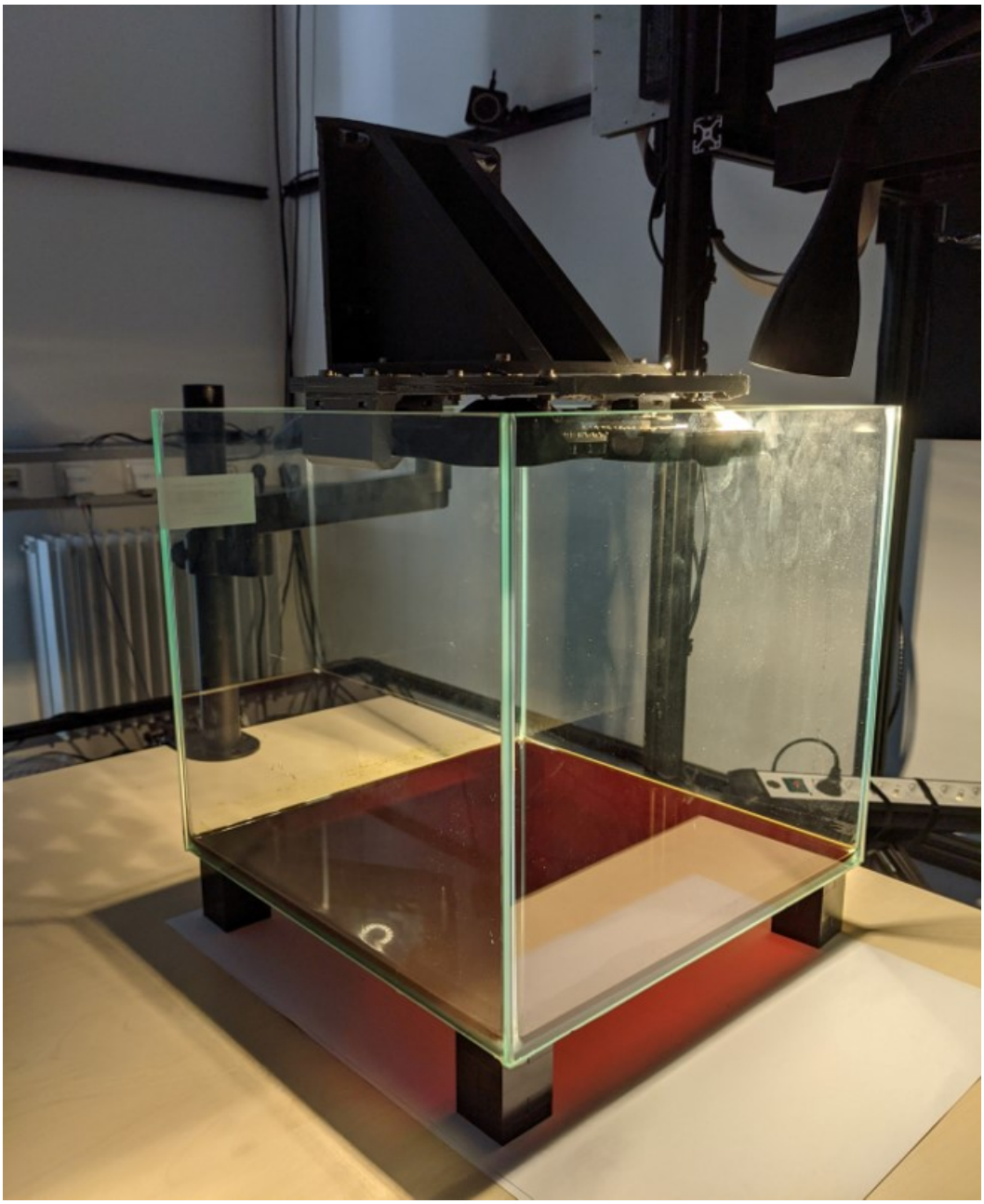

Fig. 5.11: Overall oil bath setup

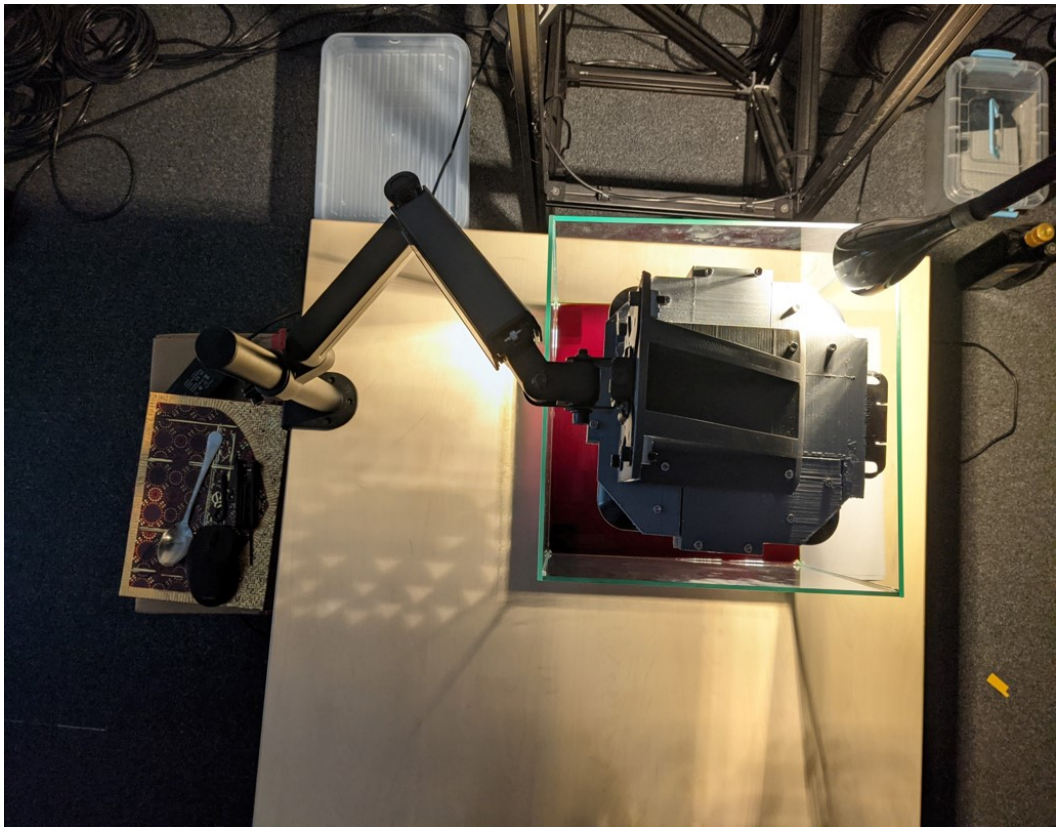

Fig. 5.12: Top view of oil bath setup

6 Conclusion

After finishing the project and observing multiple shapes using different types of rendering methods like AM and TPS, we collectively came to a conclusion that basic primitive geometric shapes(square, circle triangle, etc) are better recognized in Amplitude Modulation. Pointer movement on hand gesture using ultra leap, the feature that we call "Leap Live" showed the best results in recognition of movement and directions.

We also wrote a test script to slow down the pointer movements on the corners of a square and triangle, and concluded that corners if rendered with slow pointer movement are recognized a bit easier. The final conclusion was that for Ultraviz and oil bath experiment TPS looks better visually. However, for hand sensation AM feels better if rendering with varying intensity and frequency.

7 Future Work

The future work relating to this project would be to implement a new method of rendering. Where, the system automatically recognizes the curves, corners, edges from the varying range of dynamic input and render the same with varying intensity and speed dynamically. Also if the input is a disjoint figure, like the wifi logo as shown in fig. 5.3(b) and 5.3(b) it is better to create different control point for different disjoint lines/curves.
To include DTP method mentioned by Hajas in [Haj+20] for dynamic shapes.

8 Appendix

8.1 Source Code

AM.cs

```
using System;
using System.Collections.Generic;
using Ultrahaptics;
using Microsoft.VisualBasic.FileIO;
using System.IO;
using System.Threading;

namespace UltrahapticsShapes
{

    /**
     * Ultrahaptics Amplitude modulation class.
     * All amplitude modulation rendering takes place from this class
     * */
    public class AM
    {
        private static bool Stop = false; //boolean to stop the
    emitter
        private static AmplitudeModulationEmitter emitter;
        private static  float intensity = 1.0f;
        private static float frequency = 200.0f;

        public static void Stop_Emitter() {
            Stop = true;
        }
        public static void Render()
        {
            Stop = false;
            string file_name = Path.Combine(Environment.
    CurrentDirectory, "list.csv");
            emitter = new AmplitudeModulationEmitter();


            for (; ; )
            {
                using (TextFieldParser parser = new TextFieldParser(
    file_name))
                {
```

```
                    parser.TextFieldType = FieldType.Delimited;
                    parser.SetDelimiters(",");
                    while (!parser.EndOfData)
                    {
                        double x = 1000, y = 1000;
                        //Processing row
                        string[] fields = parser.ReadFields();
                        foreach (string field in fields)
                        {
                            //TODO: Process field
                            if (x == 1000)
                            {
                                x = double.Parse(field);
                            }
                            else if (y == 1000)
                            {
                                y = double.Parse(field);
                            }
                        }
                        Vector3 position = new Vector3((float)(x * Ultrahaptics.Units.metres), (float)(y * Ultrahaptics.Units.metres), (float)(0.20 * Ultrahaptics.Units.metres));
                        AmplitudeModulationControlPoint point = new AmplitudeModulationControlPoint(position, intensity, frequency);
                        var points = new List<AmplitudeModulationControlPoint> { point };
                        emitter.update(points);

                        //this condition will stop emitter from processing further
                        if (Stop)
                        {
                            emitter.update(new List<AmplitudeModulationControlPoint> {});

                            emitter.Dispose();
                            emitter = null;
                            Stop = false;
                            return;
                        }
                    }
                }
            }

        }
        /// <summary>
        /// Starts the live ultrahaptics rendering
        /// Note: live ultrahaptics rendering of a point always happrns in AM and is never implemented in TPS
        /// </summary>
        public static void RenderLive()
```

```
80          {
81              Stop = false;
82              emitter = new AmplitudeModulationEmitter();
83              Vector3 position = new Vector3((float)(0 * Ultrahaptics.
    Units.metres), (float)(0 * Ultrahaptics.Units.metres), (float)
    (0.20 * Ultrahaptics.Units.metres));
84              AmplitudeModulationControlPoint point = new
    AmplitudeModulationControlPoint(position, intensity, frequency);
85              var points = new List<AmplitudeModulationControlPoint> {
    point };
86              emitter.update(points);
87          }
88
89          /// <summary>
90          /// this function is to update the live rendering points
    which comes from ultraleap live rendering feature
91          /// </summary>
92          /// <param name="updated_x"> the new x coordinate</param>
93          /// <param name="updated_y"> the new y coordinate</param>
94          public static void updateLiveRenderPoint(float updated_x,
    float updated_y) {
95              if (emitter != null) {
96                  Vector3 position = new Vector3((float)(updated_x *
    Ultrahaptics.Units.metres), (float)(updated_y * Ultrahaptics.Units
    .metres), (float)(0.20 * Ultrahaptics.Units.metres));
97                  AmplitudeModulationControlPoint point = new
    AmplitudeModulationControlPoint(position, intensity, frequency);
98                  var points = new List<AmplitudeModulationControlPoint
    > { point };
99                  emitter.update(points);
100                 Console.WriteLine(updated_x + " " + updated_y);
101             }
102             if (Stop)
103             {
104                 try {
105                     emitter.update(new List<
    AmplitudeModulationControlPoint> { });
106                     emitter.Dispose();
107                     emitter = null;
108                 } catch (Exception e) {
109                     Console.WriteLine(e.Message);
110                 }
111
112                 Stop = false;
113                 return;
114             }
115
116         }
117     }
118 }
```

TPS.cs

```
using System;
using Ultrahaptics;


namespace UltrahapticsShapes
{
    /**
     * Ultrahaptics Amplitude modulation class.
     * All amplitude modulation rendering takes place from this class
     * */
    class TPS
    {
        private static TimePointStreamingEmitter _emitter;

        static uint _current = 0;
        static uint _timepoint_count = 0;

        static Vector3[] _positions;
        static float[] _intensities;
        private static bool Stop = false;
        public static void Stop_Emitter()
        {
            if (_emitter != null) {
                _emitter.stop();
                _emitter.Dispose();
                _emitter = null;
            }


        }
        public static void Render()
        {
            Stop = false;
            // Create a timepoint streaming emitter
            // Note that this automatically attempts to connect to the device, if present
            _emitter = new TimePointStreamingEmitter();
            // Inform the SDK how many control points you intend to use
            // This also calculates the resulting sample rate in Hz, which is returned to the user
            uint sample_rate = _emitter.setMaximumControlPointCount(1);
            float desired_frequency = 200.0f;
            // From here, we can establish how many timepoints there are in a single "iteration" of the cosine wave
            _timepoint_count = (uint)(sample_rate / desired_frequency);

            _positions = Utility.CSVtoVector();
```

```
            _intensities = new float[_positions.Length];

            // Populate the positions and intensities ahead of time, so that the callback is as fast as possible later
            // Modulate the intensity to be a complete cosine waveform over the full set of points.
            for (int i = 0; i < _positions.Length; i++)
            {
                float intensity = 0.5f * (1.0f - (float)Math.Cos(2.0f * Math.PI * i / _positions.Length));
                // Set a constant position of 20cm above the array
                //_positions[i] = new Vector3(0.0f, 0.0f, 0.2f);
                _intensities[i] = (1.0f);
            }

            // Set our callback to be called each time the device is ready for new points
            _emitter.setEmissionCallback(Callback, null);
            // Instruct the device to call our callback and begin emitting
            _emitter.start();
        }

        static void Callback(TimePointStreamingEmitter emitter, OutputInterval interval, TimePoint deadline, object user_obj)
        {

            foreach (var tpoint in interval)
            {
                // For each control point available at this time point...
                for (int i = 0; i < tpoint.count(); ++i)
                {
                    // Set the relevant data for this control point
                    var point = tpoint.persistentControlPoint(i);
                    point.setPosition(_positions[_current]);
                    point.setIntensity(_intensities[_current]);
                    point.enable();
                }
                // Increment the counter so that we get the next "step" next time
                _current = (_current + 1) % (uint)_positions.Length;
            }
        }
    }
}
```

UltrahapticsCS_Controller.cs

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using UltrahapticsShapes;

namespace UltrahapticsShapes
{
    class UltrahapticsCS_Controller
    {
        public static void Main(string[] args) {
            WebsocketOrchestrator.StartWebsocket();
        }
    }
}
```

Utility.cs

```
using System;
using Microsoft.VisualBasic.FileIO;
using System.IO;
using Ultrahaptics;

namespace UltrahapticsShapes
{
    public class Utility
    {

        public static Vector3[] CSVtoVector()
        {
            string file_name = Path.Combine(Environment.
    CurrentDirectory, "list.csv");
            Vector3[] positions = new Vector3[File.ReadAllLines(
    file_name).Length];
            int i = 0;
            using (TextFieldParser parser = new TextFieldParser(
    file_name))
            {
                parser.TextFieldType = FieldType.Delimited;
                parser.SetDelimiters(",");
                while (!parser.EndOfData)
                {
                    double x = 1000, y = 1000, z = 1000;
                    //Processing row
                    string[] fields = parser.ReadFields();
                    foreach (string field in fields)
                    {
                        //TODO: Process field
                        if (x == 1000)
                        {
                            x = double.Parse(field);
                        }
```

```
                        else if (y == 1000)
                        {
                            y = double.Parse(field);
                        }
                        else if (z == 1000)
                        {
                            z = double.Parse(field);
                        }
                    }
                    positions[i] = new Vector3((float)(x * Units.metres), (float)(y * Units.metres), (float)(0.2 * Units.metres));
                    i++;
                }
            }
            return positions;
        }
    }
}
```

WebsocketOrchestrator.cs

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

using WebSocketSharp;
using WebSocketSharp.Server;
using Newtonsoft.Json;

namespace UltrahapticsShapes
{

    public class Laputa : WebSocketBehavior
    {
        protected override void OnMessage(MessageEventArgs e)
        {

            if (e.Data == "BALUS") {
                Console.WriteLine(e.Data);
                Send("You are connected to ultrahaptics");
                return;
            }

            Dictionary<string,string> Json_message = JsonConvert.DeserializeObject<Dictionary<string, string>>(e.Data);

            if (Json_message["type"] == "TPS") {
                Task.Factory.StartNew(() => TPS.Render());
                Send("Check Ultrahaptics device!");
            }

```

```
            if (Json_message["type"] == "AM")
            {
                Task.Factory.StartNew(() => AM.Render());
                Send("Check Ultrahaptics device!");
            }
            if (Json_message["type"] == "leap-live-start") {

                Task.Factory.StartNew(() => AM.RenderLive());
            }

            if (Json_message["type"] == "leap-live-update")
            {

                AM.updateLiveRenderPoint(float.Parse(Json_message["X"
    ]), float.Parse(Json_message["Y"]));

            }

            if (Json_message["type"] == "stop") {
                AM.Stop_Emitter();
                TPS.Stop_Emitter();
            }


            if (Json_message["type"] == "leap-data") {

            }
        }
    }

    public class WebsocketOrchestrator
    {
        public static void StartWebsocket()
        {
            var wssv = new WebSocketServer("ws://localhost:8080");
            wssv.AddWebSocketService<Laputa>("/ultrahaptics");
            wssv.Start();
            Console.ReadKey(true);
            wssv.Stop();
        }
    }
}
```

package.json

```
{
  "name": "ultrahaptics-ejs",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "nodemon ./bin/www",
```

```
    "build": "webpack -w",
    "dev": "concurrently 'npm:build' 'npm:start' "
  },
  "dependencies": {
    "axios": "^0.21.1",
    "child_process": "^1.0.2",
    "connect-busboy": "0.0.2",
    "cookie-parser": "~1.4.4",
    "d3-array": "^2.12.1",
    "debug": "~2.6.9",
    "ejs": "~2.6.1",
    "express": "~4.16.1",
    "fs-extra": "^10.0.0",
    "http-errors": "~1.6.3",
    "morgan": "~1.9.1",
    "objects-to-csv": "^1.3.6",
    "pathologist": "^0.1.9"
  },
  "devDependencies": {
    "@babel/core": "^7.14.3",
    "@babel/preset-env": "^7.14.2",
    "babel-loader": "^8.2.2",
    "babel-minify-webpack-plugin": "^0.3.1",
    "concurrently": "^6.1.0",
    "css-loader": "^5.2.5",
    "mini-css-extract-plugin": "^1.6.0",
    "node-sass": "^6.0.0",
    "nodemon": "^2.0.7",
    "postcss-loader": "^5.3.0",
    "sass-loader": "^11.1.1",
    "webpack": "^5.37.1",
    "webpack-cli": "^4.7.0"
  }
}
```

www

```
#!/usr/bin/env node

/**
 * Module dependencies.
 */

var app = require('../app');
var debug = require('debug')('ultrahaptics-ejs:server');
var http = require('http');

/**
 * Get port from environment and store in Express.
 */

```

```
var port = normalizePort(process.env.PORT || '3000');
app.set('port', port);

/**
 * Create HTTP server.
 */

var server = http.createServer(app);

/**
 * Listen on provided port, on all network interfaces.
 */

server.listen(port);
server.on('error', onError);
server.on('listening', onListening);

/**
 * Normalize a port into a number, string, or false.
 */

function normalizePort(val) {
  var port = parseInt(val, 10);

  if (isNaN(port)) {
    // named pipe
    return val;
  }

  if (port >= 0) {
    // port number
    return port;
  }

  return false;
}

/**
 * Event listener for HTTP server "error" event.
 */

function onError(error) {
  if (error.syscall !== 'listen') {
    throw error;
  }

  var bind = typeof port === 'string'
    ? 'Pipe ' + port
    : 'Port ' + port;

  // handle specific listen errors with friendly messages
```

```
  switch (error.code) {
    case 'EACCES':
      console.error(bind + ' requires elevated privileges');
      process.exit(1);
      break;
    case 'EADDRINUSE':
      console.error(bind + ' is already in use');
      process.exit(1);
      break;
    default:
      throw error;
  }
}

/**
 * Event listener for HTTP server "listening" event.
 */

function onListening() {
  var addr = server.address();
  var bind = typeof addr === 'string'
    ? 'pipe ' + addr
    : 'port ' + addr.port;
  debug('Listening on ' + bind);
}


/**
 * Start Ultrahaptics exe
 */
//  const { exec } = require("child_process");
//  try {
//   exec("cd ../../csharp/UltrahapticsShapes/bin/x64/Debug/ &
    UltrahapticsShapes.exe",
//    function(err, data) {
//            console.log(err)
//            console.log(data)
//       })
// } catch (e) {
//   console.error(e)
// }
```

app.js

```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');
var indexRouter = require('./routes/index');
var renderRouter = require('./routes/render')
```

```
var fs = require('fs-extra');       //File System - for file
    manipulation

var app = express();


app.use(express.static(path.join(__dirname, 'public')));
// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');

//for HTML
app.engine('html', require('ejs').renderFile)

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', indexRouter);
app.use('/render', renderRouter);

// catch 404 and forward to error handler
app.use((req, res, next) => {
  next(createError(404));
});

// error handler
app.use((err, req, res, next) => {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});

module.exports = app;
```

index.js

```
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('index.html', { title: 'Express' });
});

```

```
module.exports = router;
```

render.js

```
let express = require('express');
let router = express.Router();
let renderCoordinates = require('../src/js/renderCoordinates')
/* GET users listing. */
router.post('/', function(req, res, next) {

  res.send(renderCoordinates(req.body.coordinates, req.body.
    coordinates_csv));
});

module.exports = router;
```

renderCoordinates.js

```
const ObjectsToCsv = require('objects-to-csv')
const { exec } = require("child_process");
const fs = require('fs')

const path = "../../csharp/UltrahapticsShapes/bin/x64/Debug/list.csv
    ";


function renderCoordinates (coordinates, coordinates_csv) {
  //=======================
  // const csvw = new ObjectsToCsv(coordinates);
  // csvw.toDisk("../../csharp/UltrahapticsShapes/bin/x64/Debug/
    listaaaa.csv");
  //=======================
    let csv;
    if (coordinates_csv) {
        csv = new ObjectsToCsv(coordinates);
    } else {
        let scalingFact = getScalingFactor(coordinates);
        let scaledCoordinates = coordinates.map(coordinate=>
    coordinate.map(xy=> (xy/scalingFact)- 0.04));
        scaledCoordinates.map(cord=> cord[1] = (0-cord[1])); //mirror
     x
        csv = new ObjectsToCsv(scaledCoordinates);
    }


    try {
        fs.unlinkSync(path)
        //file removed
    } catch(err) {
        console.error(err)
    }
```

```
    csv.toDisk(path);

    return "Success";
}

function getScalingFactor(coordinates) {
    return Math.max(Math.max.apply(Math,
        coordinates.map(v => v[0])),
        Math.max.apply(Math, coordinates.map(v => v[1])))/0.08
}

module.exports = renderCoordinates
```

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <style>
      #stop {
        position: absolute;
        top:0;
        right:0;
      }
      #render-button-svg,
      #clear-button-canvas,
      #render-button-canvas,
      #clear-button-leap-canvas,
      #render-button-leap-canvas{
        display: inline-block;
        vertical-align: top;
      }
      #canvas-container {
        width: 100%;
        text-align:center;
      }
      #leap-canvas, #leap-live-canvas {
        display: inline;
      }
      </style>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/
    bootstrap/3.4.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/
    jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/
    bootstrap.min.js"></script>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.1/
    jquery.min.js"></script>
<script src="http://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/
    js/toastr.min.js"></script>
```

```
<link href="http://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/
    css/toastr.min.css" rel="stylesheet"/>
    <script type="text/javascript" src="main.js"></script>
    <script src="https://js.leapmotion.com/leap-0.6.4.min.js"></
    script>
    <link type="stylesheet" src='main.css'> </link>
    </script>

  </head>

  <body onload="main()">
    <div id="bodycontainer" style="width:75%;margin:auto;">
      <div class="navbar ">
        <select id="render-type" class="form-select form-select-lg "
    aria-label=".form-select-lg example" style="margin-left:20px;">
          <option value="select" selected>-Select type of Rendering-<
    /option>
          <option value="AM">Amplitude Modulation</option>
          <option value="TPS">Time Point Streaming</option>
          <!-- <option value="CI">Custom Intelligent</option> -->
        </select>
        <button type="button" id="stop" class="btn btn-danger" style=
    "margin-right:20px;">Stop Ultrahaptics</button>
      </div>
      <ul id="navigationTabs" class="nav nav-tabs">

        <li id="navigationTab"><a href="#">File</a></li>
        <li id="navigationTab"><a href="#">Canvas</a></li>
        <li id="navigationTab"><a href="#">Leap live</a></li>
        <li id="navigationTab"><a href="#">Leap Canvas</a></li>
      </ul>

      <div class="container col-md-10 col-md-offset-1">
        <div id="file-form-tab">
          <form id="form-fileupload" class="form-signin text-center">
            <h1 class="h3 mb-3 font-weight-normal">Upload File</h1>

            <label id="tempid" for="name" class="sr-only"></label>
            <input type="file" accept=".svg,.csv" id="file_upload"
    name="file_upload" class="form-control" placeholder="Upload File">
            <div ref:svgHolder id='svg-holder'></div>
            <button class="btn btn-lg btn-primary btn-block" id="
    render-button-svg" Style="width: 150px; margin-top: 10px;">Render<
    /button>
          </form>
        </div>

        <div id="canvas-tab" style="display: none;">
          <div  ref:canvasHolder class='image-preview'>
            <h1 class="h3 mb-3 font-weight-normal" style="text-align-
    last: center;">Draw in canvas</h1>
```

```
74          <canvas ref:canvas id='shape-canvas' width='500' height
    ='500'
75            style="border: 2px solid #000000;
76                background-color: gray;
77                margin: 0 auto;
78                display: table;
79                width: 500;
80                height: 500;">
81          </canvas>
82        </div>
83        <div class="row">
84          <div class="col-sm-12 text-center">
85            <button class="btn btn-primary btn-md center-block" id=
    "clear-button-canvas" Style="width: 150px;">Clear Canvas</button>
86            <button class="btn btn-primary btn-md center-block" id=
    "render-button-canvas" Style="width: 150px;">Render Canvas</button
    >
87          </div>
88        </div>
89      </div>
90
91      <div id="leap-live-canvas-tab" style="display: none;">
92          <div  ref:canvasHolder class='image-preview'>
93            <h1 class="h3 mb-3 font-weight-normal" style="text-
    align-last: center;">Draw in canvas using leap motion</h1>
94            <canvas ref:canvas id='leaplivecanvas' width='500'
    height='500'
95              style="border: 2px solid #000000;
96                  background-color: white;
97                  margin: 0 auto;
98                  display: table;
99                  width: 500;
100                 height: 500;">
101           </canvas>
102         </div>
103         <div class="row">
104           <div class="col-sm-12 text-center">
105             <button class="btn btn-primary btn-md center-block"
    id="start-leap-button" Style="width: 150px;">Start Leap</button>
106           </div>
107         </div>
108     </div>
109
110     <div id="leap-canvas-tab" style="display: none;">
111       <div  ref:canvasHolder class='image-preview'>
112         <h1 class="h3 mb-3 font-weight-normal" style="text-align-
    last: center;">Draw in canvas using leap motion</h1>
113         <canvas ref:canvas id='leapcanvas' width='500' height
    ='500'
114           style="border: 2px solid #000000;
115               background-color: white;
```

```
                    margin: 0 auto;
                    display: table;
                    width: 500;
                    height: 500;">
              </canvas>
            </div>
            <div class="row">
              <div class="col-sm-12 text-center">
                <button class="btn btn-primary btn-md center-block" id="clear-button-leap-canvas" Style="width: 150px;">Clear Canvas</button>
                <button class="btn btn-primary btn-md center-block" id="render-button-leap-canvas" Style="width: 150px;">Render Canvas</button>
              </div>
            </div>
          </div>
        </div>

    </div>
    <div id="toast"></div>
  </body>
</html>
```

startApp.js

```
import pathologize from './pathologize';
import pathsToCoords from './pathsToCoords';
import axios from 'axios';
import leap from './leap';
import leap_canvas from './leapCanvas.js'
import canvas_script from './canvas_script';
import cs_websocket from './cs_websocket'

/**
 * this data is used while converting the svg to the coordinate points
 * NOTE: the scaling of the coordinates to render on ultrahaptics device
 * for size adjustment is done in renderCoordinates.js
 *
 * we recommend to change only number of points here.
 **/
var data= {
    scale:1,
    numPoints:1000, //number of points to be generated from svg to coordinates
    translateX:1,
    translateY:1
}

```

```
//as soon as the application starts this function controls the
//html dom for all the functions.
export default function startApp () {

    window.ocs_websocket = cs_websocket();
    window.coordinates;


    /**As soon as the file is uploaded in the first tab the
     * control comes here to handle the uploaded file and process it
     * a) if the file is svg, transform it to the canvas and get all
    the coordinates
     * b) if the file is csv just extract the coordinates from it
     *
     * The  csv coordinates should be scaled according to
    ultrahaptics,
     * this function do not process the coordinates from csv
     **/
    var handleFileUpload = function (e) {
        debugger;
        window.coordinates = [];
        let file = this.files[0];
        const reader = new FileReader();

        if (file.type == 'application/vnd.ms-excel') {
            reader.onload = function() {
                var allTextLines = reader.result.split(/\r\n|\n/);

                //below for loop converts the csv file from string
    form to array of coordinates
                for (var i=0; i<allTextLines.length; i++) {
                    var data = allTextLines[i].split(',');
                    if (allTextLines[i] == '') continue; //empty
    lines in csv file will nullify here
                        var tarr = [];
                        for (var j=0; j<data.length; j++) {
                            tarr.push(data[j]);
                        }
                        window.coordinates.push(tarr);
                }

            };
        }
        if (file.type === 'image/svg+xml') {
            reader.onload = function() {
                //pathologize: extracts the path elements from the
    svg
                let path = pathologize(reader.result)
                renderSVGInHTML(path);
            };
        }
```

```
69
70          reader.readAsText(file);
71      }
72
73      //renders svg in html for preview and visuals
74      var renderSVGInHTML = function (pathsOnly) {
75          // document.body.appendChild(pathsOnly)
76          document.getElementById('svg-holder').innerHTML = pathsOnly
77          var paths = document.getElementById('svg-holder').
    getElementsByTagName('path');
78
79          //converts the path elements to coordinates
80          window.coordinates=pathsToCoords(paths, data.scale, data.
    numPoints, data.translateX, data.translateY)
81      }
82
83
84      /**
85       * as soon as you click the render button on
86       * any tab the coordinates are sent to the backend
87       *  via axios request from this function.
88       *
89       * This function, sends the data to nodejs server and the node js
     function
90       *  writes the coordinates  data in a file.
91       * In the callback this function informs the ultrahaptics via
92       * websocket to render the updated coordinates from the file.
93       *
94       * @param {*} coordinates array of arrays i.e x,y coordinates
95       */
96      var handleRenderButtonCLicked = function(coordinates) {
97          debugger;
98          let render_type = $('#render-type option:selected').val()
99          if (render_type == 'select') {
100
101             toastr["error"]("Please select type of rendering, from
    top left drop down.",
102              "Ultrahaptics")
103
104         } else {
105             let coordinates_csv = false;
106             if (document.querySelector('#file_upload').files[0] &&
107             document.querySelector('#file_upload').files[0].type == '
    application/vnd.ms-excel')  {
108                 coordinates_csv = true;
109             }
110             axios({
111                 method: 'post',
112                 url: '/render',
113                 data: {
114                   coordinates: coordinates,
```

```
                coordinates_csv: coordinates_csv
              }
            }).then(function (response) {
                console.log(response);
                if (response.data == 'Success') {
                  let render_type = $('#render-type option:selected
').val()
                  window.ocs_websocket ? window.ocs_websocket.
send_message({'type': render_type}) :null;
                }
            });
    }
}

/**
 * Whenever a new tab in UI is clicked the control comes here
 * @param {*} event
 */
var handleTabChanged = function(event) {

    if (this.innerText =="File") {
        $("#file-form-tab").show();
        $("#canvas-tab").hide();
        $("#leap-live-canvas-tab").hide();
        $("#leap-canvas-tab").hide();
        handleStopClicked();
    } else if (this.innerText =="Canvas") {
        $("#file-form-tab").hide();
        $("#canvas-tab").show();
        $("#leap-live-canvas-tab").hide();
        $("#leap-canvas-tab").hide();

        //canvas was showing random points after opening. this
will clean the canvas
        var canvas = document.getElementById('shape-canvas');
        var ctx = canvas.getContext('2d');
        ctx.fillStyle = "#FFFFFF";
        ctx.clearRect(0, 0, ctx.width, ctx.height);
        ctx.fillRect(0, 0, ctx.canvas.width, ctx.canvas.height);
        ctx.beginPath();
        window.canvas_coordinates = []
        handleStopClicked();
    } else if (this.innerText =="Leap live") {
        $("#file-form-tab").hide();
        $("#canvas-tab").hide();
        $("#leap-live-canvas-tab").show();
        $("#leap-canvas-tab").hide();
        handleStopClicked();

        setTimeout(function(){ toastr["success"]('Leap Live Ready
', "Leap");
```

```
            leap(); }, 2000);
        } else if (this.innerText == "Leap Canvas") {
            $("#file-form-tab").hide();
            $("#canvas-tab").hide();
            $("#leap-live-canvas-tab").hide();
            $("#leap-canvas-tab").show();
            handleStopClicked();

            setTimeout(function(){ toastr["success"]('Leap Canvas
Ready', "Leap");
            leap_canvas(); }, 2000);
        }
    }
    var handleRenderCanvasButtonCLicked = function() {
        handleRenderButtonCLicked(window.canvas_coordinates)
    }

    var handleRenderLeapCanvasButtonCLicked = function() {
        handleRenderButtonCLicked(window.leap_canvas_coordinates)
    }

    var handleRenderSVGButtonCLicked = function() {
        debugger;
        handleRenderButtonCLicked(window.coordinates)
    }

    //Stops the ultrahaptics device by communicating over websocket
    var handleStopClicked = function() {
        window.ocs_websocket ? window.ocs_websocket.send_message({"
type":"stop"}) : null;
    }

    canvas_script();

    document.querySelector('#file_upload').addEventListener("change",
     handleFileUpload)
    document.getElementById('render-button-svg').addEventListener('
click', handleRenderSVGButtonCLicked)
    document.getElementById('render-button-canvas').addEventListener(
'click', handleRenderCanvasButtonCLicked)
    document.getElementById('render-button-leap-canvas').
addEventListener('click', handleRenderLeapCanvasButtonCLicked)
    document.getElementById('stop').addEventListener('click',
handleStopClicked);
    document.querySelectorAll('#navigationTab').forEach(item=> {
        item.addEventListener("click", handleTabChanged);
    })



    /**
```

```
     * Settings for thetoastr library which is
     *  responsible for showing the layover warning or success
    messages
     */
    toastr.options = {
        "closeButton": false,
        "debug": false,
        "newestOnTop": false,
        "progressBar": false,
        "positionClass": "toast-bottom-right",
        "preventDuplicates": false,
        "onclick": null,
        "showDuration": "500",
        "hideDuration": "1000",
        "timeOut": "5000",
        "extendedTimeOut": "1000",
        "showEasing": "swing",
        "hideEasing": "linear",
        "showMethod": "fadeIn",
        "hideMethod": "fadeOut"
      }

}
```

pathologize.js

```
//Thank you Rich!! https://pathologist.surge.sh/
import * as pathologist  from 'pathologist';

export default function pathologize (original) {

  //handles issues with pathologist not parsing text and style
    elements
  const reText = /<text[\s\S]*?<\/text>/g;
  const reStyle = /<style[\s\S]*?<\/style>/g;
  const removedText = original.replace(reText, '');
  const removedStyle = removedText.replace(reStyle, '');

  try {
    //transforms the svg to the coordinates for the canvas
    const pathologized = pathologist.transform(removedStyle);
    return pathologized;
  } catch (e)  {
    return original;
  }
}
```

polygonize.js

```
import { range } from 'd3-array';

```

```
export default function polygonize (path, numPoints, scale,
    translateX, translateY) {

  const length = path.getTotalLength();

  return range(numPoints).map(function(i) {
    const point = path.getPointAtLength(length * i / numPoints);
    return [point.x * scale + translateX, point.y * scale +
    translateY];
  });
}
```

pathsToCoords.js

```
import polygonize from './polygonize';
import getTotalLengthAllPaths from './getTotalLengthAllPaths';


export default function pathsToCoords ( paths, scale, numPoints,
    translateX, translateY ) {
  const totalLengthAllPaths = getTotalLengthAllPaths( paths );

  let runningPointsTotal = 0;
  const separatePathsCoordsCollection = Array.from(paths).reduce((
    prev, item, index) => {
    let pointsForPath;
    if (index + 1 === paths.length) {
      //ensures that the total number of points = the actual
    requested number (because using rounding)
      pointsForPath = numPoints - runningPointsTotal;
    } else {
      pointsForPath = Math.round(numPoints * item.getTotalLength() /
    totalLengthAllPaths);
      runningPointsTotal += pointsForPath;
    }
    return [...prev, ...polygonize(item, pointsForPath, scale,
    translateX, translateY)];
  }, []);
  return separatePathsCoordsCollection;
}
```

leap.js

```
import cs_websocket from "./cs_websocket";

export default function leap() {

  var controller = new Leap.Controller({enableGestures: true,
    frameEventName: 'deviceFrame'});
  var ctrl = new Leap.Controller({enableGestures: true,
    frameEventName: 'deviceFrame'});
```

```
var canvas = document.getElementById('leaplivecanvas');
var ctx = canvas.getContext('2d');

reset_canvas();

window.leap_live_canvas_coordinates = []

var centerX = 0;
var centerY = 0;

sizing();

function reset_canvas(){
  ctx.fillStyle = "#FFFFFF";
  ctx.clearRect(0, 0, ctx.width, ctx.height);
  ctx.fillRect(0, 0, ctx.canvas.width, ctx.canvas.height);
  ctx.beginPath();
  window.leap_live_canvas_coordinates = []
  controller.connect();
}

$(window).resize(function() {
  sizing();
});

function sizing() {
  centerX = canvas.width / 2;
  centerY = canvas.height / 2;
}

controller.on('gesture', function (gesture) {
  if (gesture.type == 'keyTap') {
    toastr["info"]('Start Drawing', "Leap");

    window.ocs_websocket ? window.ocs_websocket.send_message({type:
  'leap-live-start',X:0, Y:0}) :null;


    controller.disconnect();
    ctrl.connect();
    ctrl.on('frame', function(frame) {
      if (frame.fingers[0]) {
        var x = frame.fingers[1].tipPosition[0];
        var y = frame.fingers[1].tipPosition[1];

        var scalingfactor = 500/0.16
        // console.log(x + centerX, );
        draw(x + centerX, canvas.height - y);
        window.ocs_websocket ? window.ocs_websocket.send_message({
  type:'leap-live-update',X:(x+centerX)/scalingfactor -0.08, Y: (0 -
  (canvas.height - y)/scalingfactor)+0.08})  :null;
```

```
        }
      });

      ctrl.on('gesture', function (gesture) {
        if (gesture.type == 'keyTap') {
          toastr["info"]('Drawing stopped.', "Leap");
          window.ocs_websocket ? window.ocs_websocket.send_message({"
    type":"stop"}) : null;
          window.leap_live_canvas_coordinates = [];
          ctrl.disconnect();
        }
      });
    }
  });


  function draw(x, y, radius) {

    ctx.clearRect(0, 0, canvas.width, canvas.height);

    if (typeof radius === 'undefined') {
      radius = 4;
    }
    else {
      radius = radius < 4 ? 4 : radius;
    }

    ctx.fillStyle = '#000000';
    ctx.strokeStyle = '#000000';
    ctx.beginPath();
    ctx.arc(x, y, radius, 0, Math.PI * 2, true);
    ctx.fill();

    // cs_websocket.send({type:'leap live',X:x, Y:y})

    // console.log(x + " " + y + "\n")

  }

  toastr["info"]('KeyTap gesture to start drawing!', "Leap")

  controller.on('deviceAttached', function (gesture) {
    toastr["info"]('Leap Motion device is ready!', "Leap")
  });

  controller.connect();

  document.getElementById('start-leap-button').addEventListener('
    click', reset_canvas);

}
```

leapCanvas.js

```
export default function leap_canvas() {

  var controller = new Leap.Controller({enableGestures: true,
    frameEventName: 'deviceFrame'});
  var ctrl = new Leap.Controller({enableGestures: true,
    frameEventName: 'deviceFrame'});
  var canvas = document.getElementById('leapcanvas');
  var context = canvas.getContext('2d');

  reset_canvas();

  window.leap_canvas_coordinates = []

  var centerX = 0;
  var centerY = 0;

  sizing();

  function reset_canvas(){
    context.fillStyle = "#FFFFFF";
    context.clearRect(0, 0, context.width, context.height);
    context.fillRect(0, 0, context.canvas.width, context.canvas.
    height);
    context.beginPath();
    window.leap_canvas_coordinates = []
    controller.connect();
  }

  controller.on('gesture', function (gesture) {
    if (gesture.type == 'keyTap') {
      fadeOut('Start Drawing!');
      controller.disconnect();
      ctrl.connect();
      ctrl.on('frame', function(frame) {
        if (frame.fingers[0]) {
          var x = frame.fingers[1].tipPosition[0];
          var y = frame.fingers[1].tipPosition[1];
          draw(x + centerX, canvas.height - y);
        }
      });

      ctrl.on('gesture', function (gesture) {
        if (gesture.type == 'keyTap') {
          toastr["info"]('Drawing stopped.', "Leap")
          ctrl.disconnect();
        }
      });
    }
  });

```

```
48  $(window).resize(function() {
49    sizing();
50    draw(centerX, centerY);
51  });
52
53  function sizing() {
54    centerX = canvas.width / 2;
55    centerY = canvas.height / 2;
56  }
57
58  function draw(x, y, radius) {
59    if (typeof radius === 'undefined') {
60        radius = 4;
61    }
62    else {
63        radius = radius < 4 ? 4 : radius;
64    }
65
66    context.strokeStyle = '#000000';
67    context.fillStyle = '#000000';
68    context.lineWidth = 5;
69    context.beginPath();
70    context.arc(x, y, radius, 0, Math.PI * 2, false);
71    context.fill();
72    window.leap_canvas_coordinates.push([x,y])
73  }
74
75  function fadeOut(text) {
76    var alpha = 1.0,   // full opacity
77        interval = setInterval(function () {
78            canvas.width = canvas.width; // Clears the canvas
79            context.fillStyle = "#000000" + alpha + ")";
80            context.font = "italic 20pt Arial";
81            context.fillText(text, 50, 50);
82            alpha = alpha - 0.05; // decrease opacity (fade out)
83            if (alpha < 0) {
84                canvas.width = canvas.width;
85                clearInterval(interval);
86            }
87        }, 50);
88  }
89
90  toastr["info"]('KeyTap gesture to start drawing!', "Leap")
91
92  controller.connect();
93
94  document.getElementById('clear-button-leap-canvas').
      addEventListener('click', reset_canvas);
95
96}
```

getTotalLengthAllPaths.js

```
export default function getTotalLengthAllPaths (paths) {
  return Array.from(paths).reduce((prev, curr) => {
    return prev + curr.getTotalLength();
  }, 0);
}
```

cs_websocket.js

```

export default function cs_websocket() {
    let ws = new WebSocket('ws://localhost:8080/ultrahaptics');
    ws.onopen=()=> {
        ws.send("BALUS");
    }
    ws.onmessage = (evt)=>  {

        var received_msg = evt.data;
        // alert("Message received = "+ received_msg);
        toastr["success"](received_msg, "Ultrahaptics")
    };
    ws.onclose =()=> {

        setTimeout(function() {window.ocs_websocket = cs_websocket()
    ;}, 5000)
    };

    return {
        send_message : (data) => {
            if(ws) {
                ws.send(JSON.stringify(data))
            }

        },

    }
};
```

canvas_script.js

```


export default function canvas_script () {
    //canvas functions
  var canvas = document.getElementById('shape-canvas');
  var ctx = canvas.getContext('2d');
  reset_canvas();
  // CODE FOR MOUSE AND DRAWING:

  var pos = { x: 0, y: 0 }; // last known mouse position
  window.canvas_coordinates = []
```

```
  // mouse event liseners:
  document.addEventListener('mousemove', draw);
  document.addEventListener('mousedown', setPosition);
  document.addEventListener('mouseenter', setPosition);
  $("#clear_button").click(reset_canvas);

  // new position from mouse event
  function setPosition(e) {
    var rect = canvas.getBoundingClientRect();
    pos.x = e.clientX - rect.left;
    pos.y = e.clientY - rect.top;
  }

  // clear the drawing
  function reset_canvas() {
      ctx.fillStyle = "#FFFFFF";
      ctx.clearRect(0, 0, ctx.width, ctx.height);
      ctx.fillRect(0, 0, ctx.canvas.width, ctx.canvas.height);
      ctx.beginPath();
      window.canvas_coordinates = []
  }

  // allow user to draw on the canvas by connecting subsequent mouse
    points with a line
  function draw(e) {
    // mouse left button must be pressed
    if (e.buttons !== 1) return;

    ctx.lineWidth = 5;
    ctx.lineCap = 'round';
    ctx.strokeStyle = '#000000';

    // TODO: implement the drawing functionality
    ctx.moveTo(pos.x, pos.y);
    setPosition(e);
    ctx.lineTo(pos.x, pos.y);
    ctx.stroke();
    window.canvas_coordinates.push([pos.x,pos.y])
  }

  document.getElementById('clear-button-canvas').addEventListener('
    click', reset_canvas);

}
```

main.js

```
import startApp from './js/startApp'
let main = function() {
    startApp()
}
```

```
5 window.main = main
```

Bibliography

[Am] *What is Amplitude Modulation? – Ultrahaptics Developer Information* (cit. on p. 3).

[Bab] *Babel · The compiler for next generation JavaScript* (cit. on p. 10).

[Fri+18] William Frier, Damien Ablart, Jamie Chilles, et al. „Using spatiotemporal modulation to draw tactile patterns in mid-air“. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 10893 LNCS (2018), pp. 270–281 (cit. on p. 3).

[Haj+20] Daniel Hajas, Dario Pittera, Antony Nasce, Orestis Georgiou, and Marianna Obrist. „Mid-Air Haptic Rendering of 2D Geometric Shapes with a Dynamic Tactile Pointer“. In: *IEEE Transactions on Haptics* 13.4 (2020), pp. 806–817 (cit. on p. 25).

[ITS08] Takayuki Iwamoto, Mari Tatezono, and Hiroyuki Shinoda. *Non-contact method for producing tactile sensation using airborne ultrasound*. 2008, pp. 504–513 (cit. on p. 1).

[Lea] *Controller — Leap Motion JavaScript SDK v3.2 Beta documentation* (cit. on p. 7).

[Oil] *How does Ultrahaptics technology work? – Ultrahaptics Developer Information* (cit. on p. 18).

[Spo] *GitHub - spotify/coordinator: A visual interface for turning an SVG into XY coördinates.* (Cit. on p. 10).

[Tps] *What is Time Point Streaming? – Ultrahaptics Developer Information* (cit. on p. 4).

[Ul] *Digital worlds that feel human | Ultraleap* (cit. on pp. 1, 13).

[Ulta] *ultraleap-labs/Ultraviz at master · ultraleap/ultraleap-labs · GitHub* (cit. on p. 13).

[Ultb] *Using Ultrahaptics APIs: Amplitude Modulation and Time Point Streaming – Ultrahaptics Developer Information* (cit. on p. 11).

[Wei+13] Frank Weichert, Daniel Bachmann, Bartholomäus Rudak, and Denis Fisseler. „Analysis of the Accuracy and Robustness of the Leap Motion Controller“. In: *Sensors* 13.5 (2013), pp. 6380–6393 (cit. on p. 7).

List of Figures

Declaration

I declare that this project, was solely undertaken by myself. All sections of the report that use or describe an concept developed by another author are referenced.

Bayreuth, September 27, 2021

Anuj Sharma,
Akshaya Kumar Krishnappa Ramakrishna