

**SCHOOL OF COMPUTING  
UNIVERSITY OF TEESSIDE  
MIDDLESBROUGH  
TS1 3BA**

**Touch Surface Technology**

**Computer Games Science**

**James William Fletcher**

**4<sup>th</sup> April**

**Supervisor: Jay Chapman**

**Second Reader:**

## **ABSTRACT**

Touch surfaces are surfaces that can behave like a touch screen, without the requirement of a unique/fixed screen/surface. Although not required, removing the limitation of a unique/fixed stylus is preferred.

The problem with touch surface technology is that it is currently achieved using digital cameras and computer vision. Computer Vision techniques cannot detect the force at which a user is touching the surface from the data provided by digital cameras in the form of images and it is computationally slow to perform object tracking using a digital camera.

What I achieve is a solution to these problems of performance scalability, high computational cost and inability to detect factors such as touch force. My proposed idea is that piezo electric contact microphones are used to pick up the vibrations on a surface caused by an object touching it; this is in order to work out the position the touch vibration originated.

The project is implemented in C++ on a windows platform, although open source dependencies have been used to ensure full portability amongst a wide variety of platforms.

The project uses current day standards in computer audio, 16 bit sampling at a rate of 44100 Hz.

This project takes a lot of computational expense away from the input and allows more computational expense to be consumed by running applications; this allows for greater visual quality and simulation in the applications that use touch surface technology.

## **ACKNOWLEDGEMENTS**

It is a pleasure to thank those who made this possible, particularly my project supervisor Jay Chapman who supported me throughout the project and whom I acquired substantial knowledge of presentations from.

Special thanks to Gary Quinn of the motion capture studio at Teesside University who was very helpful offering advice and help acquiring resources, also to the friendly community over at KVR Audio for expertise and patience.

I offer my regards to all those that supported me along the way, particularly wang-eye cat and little bear who comforted me to sleep after late nights, peter the reading pigeon, raffles the unicycle riding dog, carrot rabbit and the mirror for great conversation and company as I worked.

Lastly I would like to thank my parents for their life long support and understanding in dedicated endeavours of work, creativity, exploration and experimentation.

# CONTENTS

<u>ABSTRACT.....</u>	<u>I</u>
<u>ACKNOWLEDGEMENTS.....</u>	<u>II</u>
<u>1 INTRODUCTION.....</u>	<u>5</u>
<u>2 METHODOLOGY.....</u>	<u>7</u>
<u>3 HARDWARE RESEARCH.....</u>	<u>9</u>
<u>3.1 Microphones.....</u>	<u>9</u>
<u>3.2 Soundcards.....</u>	<u>11</u>
<u>3.3 DAQ.....</u>	<u>12</u>
<u>3.4 Audio Cable.....</u>	<u>13</u>
<u>4 SOFTWARE RESEARCH.....</u>	<u>14</u>
<u>4.1 Third Party Libraries.....</u>	<u>14</u>
<u>4.2 Digital Signal Processing.....</u>	<u>15</u>
<u>4.2.1 FIR &amp; IIR Filters.....</u>	<u>15</u>
<u>4.2.2 Fourier Transform.....</u>	<u>16</u>
<u>4.2.3 Intersample Interpolation.....</u>	<u>17</u>
<u>5 MATERIAL RESEARCH.....</u>	<u>19</u>
<u>5.1 Surface Material Properties.....</u>	<u>19</u>
<u>5.2 Material Surface Texture &amp; Touching Stylus.....</u>	<u>20</u>
<u>5.3 Attachment of Contact Microphones.....</u>	<u>21</u>
<u>6 ANALYSIS.....</u>	<u>22</u>
<u>6.1 Target Audience.....</u>	<u>22</u>
<u>6.2 Computer Vision Approaches.....</u>	<u>26</u>
<u>6.3 Performance Analysis.....</u>	<u>29</u>
<u>6.4 Multi-Touch Detection.....</u>	<u>31</u>
<u>6.5 Double Buffering.....</u>	<u>34</u>
<u>6.6 Trilateration.....</u>	<u>35</u>
<u>7 DESIGN.....</u>	<u>39</u>
<u>7.1 Hardware.....</u>	<u>39</u>
<u>7.2 Pipeline.....</u>	<u>39</u>
<u>7.3 Binary Double Buffer.....</u>	<u>40</u>
<u>7.4 Touch Detector.....</u>	<u>44</u>

7.5 Trilaterator.....	46
7.5.1 Calibrator.....	47
7.6 User Interface.....	47
<b>8 IMPLEMENTATION.....</b>	<b>48</b>
8.1 Intersample Interpolation.....	48
8.2 Prototype Implementation.....	49
<b>9 PROFILING &amp; TESTING.....</b>	<b>51</b>
9.1 Binary Double Buffer Performance Analysis.....	51
9.2 Identification of Pipeline Bottlenecks.....	54
<b>10 FUTURE OBJECTIVES.....</b>	<b>56</b>
10.1 OpenAS.....	56
10.2 Dedicated Hardware.....	56
10.3 Plugin-Development.....	56
<b>11 RECOMMENDATIONS.....</b>	<b>57</b>
11.1 Multi-Threading.....	57
11.2 Parallel Computing.....	57
11.3 Time to Digital Converter.....	57
11.4 Laser Microphones.....	57
<b>12 CONCLUSION.....</b>	<b>58</b>
<b>REFERENCES.....</b>	<b>59</b>
<b>APPENDIX A SPECIFICATION.....</b>	<b>62</b>
<b>APPENDIX B PIPELINE DIAGRAMS.....</b>	<b>63</b>
<b>APPENDIX C INTERSAMPLE INTERPOLATION FUNCTION.....</b>	<b>64</b>
<b>APPENDIX D BINARY DOUBLE BUFFER CLASS.....</b>	<b>71</b>
<b>APPENDIX F PROJECT DIARIES.....</b>	<b>80</b>
<b>APPENDIX G QUESTIONNAIRES.....</b>	<b>87</b>

# 1 INTRODUCTION

Over the decades of computing Human Computer Interaction (HCI) has been so influential that it now has own field in computer science, we have seen all kinds of abstract ideas for HCI, most notably the mouse, and since the field has grown into a wide variety of experimental and practical applications such as the Brain Computer Interface (BCI) which monitors brain activity using non-intrusive sensors that sit around the forehead; but even more notable is touch screen technology which in recent years has been pioneered by Steve Jobs at Apple with the iPod, iPhone and iPad. The first touch sensor was developed by Doctor Sam Hurst in 1971, he called it the Elograph and it defines the first ever touch surface created as it lacked the transparency it would have required to have been attached to the surface of a monitor screen. In 1974 Hurst developed a transparent model and then in 1977 Hurst went on to develop five-wire resistive touch technology which is the most widely used touch screen technology today.

The scope of this project is to construct the hardware and software of a fully working touch surface prototype that is capable of detecting touch force, position and gestures on a wide variety of surfaces that carry vibrations; this is with the aim to produce a less computationally expensive solution than the existing computer vision solutions on the market.

The targets can be broken down into four points;

- Analysis of target audience and identification of consumer base.
- Development of a cheap hardware solution.
- Development of operating system software to interface with hardware.
- Development of a prototype to demonstrate the technology.

In the following chapters I describe the analysis, research, design, implementation and testing. This project is based on an idea I independently discovered and have no knowledge of anyone having implemented anything

similar in the same context prior, so please read on with respect that most methods were independently discovered through perseverance.

## 2 METHODOLOGY

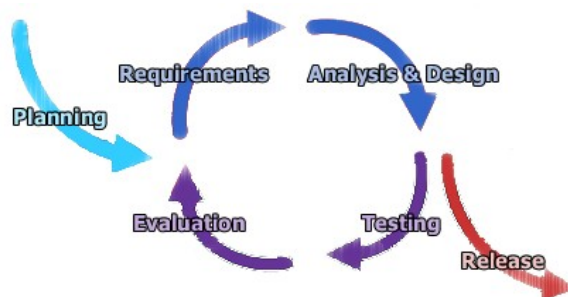
The iterative scope of the methodology can be broken down into eight fundamental questions;

- Who is my target audience, what do they make of the possibilities and pros/cons of this solution and most importantly; do they feel it will be useful?
- What hardware can I build the prototype from that offers low price and acceptable quality audio processing?
- What is the best method of interfacing between hardware and software?
- What is the best method of detecting that a touch impact has occurred, and filtering out any background noise that may lead to a false detection?
- What is the most efficient method of detecting the origin of a vibration from the information available from four contact microphones?
- What is the most accurate and computationally inexpensive method to detect multiple impacts of different objects at the same time?
- From the information available what input 'gestures' can be detected?
- What configurations can I expose to the customer in the form of a user interface to improve performance and accuracy for specific applications of the technology?

While answering these questions, the results of one question could affect the results of a previously answered question. For example, once I have

performed a target audience analysis it may become apparent that the solution needs to be achieved under a particular budget otherwise potentially no one would buy; but it may also be apparent that the audience requires a particular accuracy from the product; if when I come to pick the hardware within the desired price range of the target audience and the accuracy produced by the chosen hardware is insufficient for the price range they specified I would have to go back and perform a re-analysis of the target audience to attain what can be done to resolve this problem.

After looking at a range of methodologies; narrowed down to Spiral, Agile, Waterfall and Iterative; It was found that the Iterative approach was best suited. This is because Agile although it is very customer oriented and geared towards a quick development cycle and release, it is also designed particularly around teams of multiple people, and thus was insufficient in particular areas and elaborated in others such as the treatment of employees and interaction with customers. Waterfall was too linear to be practical for a software development process in that it doesn't allow re-iteration of previous stages. The spiral model was particularly attractive in its more realistic and natural approach to software development, however the methodology is better suited for higher risk projects in which this project has no real financial risk.



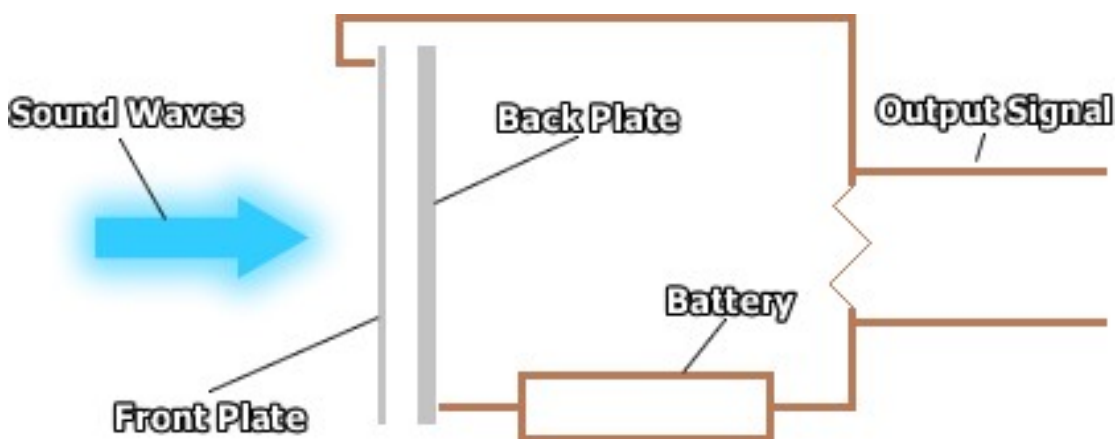
The iterative / incremental development model defines that projects are laid out in a linear sequence of stages although if required re-iteration on previous stages is possible. The implementation guidelines are also better suited towards object oriented programming, which goes hand in hand with the re-iterative nature of the model; making it easier to re-iterate over particular modules and patches.

### 3 HARDWARE RESEARCH

#### 3.1 Microphones

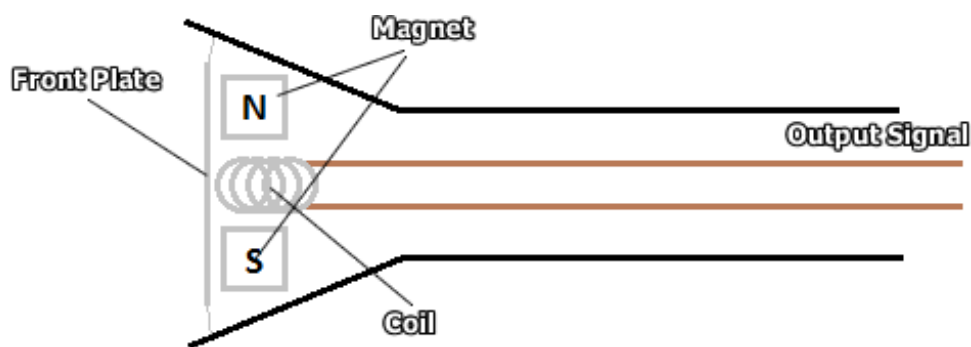
One of the fundamental components of hardware is the microphone, without it there would be no way of converting vibrations into analog signals. The first microphone was invented by Emile Berliner for use as a telephone voice transmitter. There are many ways to convert vibrations into electronic signals so the following paragraphs only cover the relevant methods.

The **Condenser Microphone** is very sensitive method of measuring vibrations, it consists of a front plate and a back plate and a current is transmitted between the plates. The front plate is made of a very light weight metal so that when a vibration hits the front plate it vibrates along with it, the vibration of the front plate causes the distance between the two plates to vary and thus changes the capacitance of the current between the two plates.



These types of microphone need to be individually powered by an external power source.

**Dynamic Microphones** work in a similar way to condenser microphones, sound waves hit the front plate and cause a small coil to move between a fixed magnetic field which produces a varying current in the coil; they do not require any external source of power and although not quite as sensitive as condenser microphones they are a lot cheaper to produce.



**Piezoelectric Microphones** use a phenomenon called piezoelectricity, this is where some materials exhibit the ability to produce electrical current when subjected to pressure; typically Lead-zirconium titanate (PZT) is used although there is a wide range of piezoelectric materials. Piezoelectric microphones produce high distortion which makes them insufficient for high quality recording however they are inexpensive due to their simple construction. It's best to use a pre-amplifier with any piezoelectric microphone that is attached to a long piece of audio cable otherwise the resistance of the wire could greatly affect the amplitude received by the sound card.

**Fibre-Optic Microphones** produce electrical signals from changes in light intensity, light from a laser travels through an optical fibre to illuminate the front plate, when the front plate vibrates the intensity of the light it reflects changes. The light goes through a photo detector and gets converted into an electrical current. Fibre Optic Microphones possess a high frequency range, they are robust designs and do not react to electrical, electrostatic, magnetic or radioactive fields making them safe to use in magnetic resonance imaging.

**Laser Microphones** fire a small laser beam at a surface and monitor the minute vibrations of the surface through the displacement of the returned beam. The advantage is that they can be used to monitor surface vibrations remotely and as such these are generally associated to techniques of remotely eavesdropping on conversations of some distance from the eavesdropper. Laser Microphones are inexpensive and relatively simple to build.

Microphones come in many different shapes and sizes, particularly depending on their application. For example a TV presenter uses Lavalier microphones which are the discreet microphones they wear on their bodies; contact microphones are design to pick up surface vibrations, this is particularly useful in acoustics where the sound produced by an instrument needs to be converted into an electrical signal and noise-cancelling microphones are designed for noisy environments.

### **3.2 Soundcards**

A microphone produces an analog signal which needs to be converted into a digital signal before it can be processed by a computer. This requires an analog to digital converter (ADC), which is partly what a computer sound card is. Some microphones come with an ADC built in; usually these are sold as USB Microphones.

The ADC defines the sample rate or frequency at which the signal is sampled and converted to a digital signal and it also defines how many amplitude bits are quantized, typically 16 bit quantization at a sampling frequency of 44.1 KHz is used as this is the widely adopted CD quality audio standard.

A problem with quantizing an analog signal is that information gets lost, often larger amplitudes are under sampled and the ends get cut off; to avoid this problem a preamplifier or potentiometer can be used to scale the range of the analog signal before it enters the ADC.

It's best to use a Sound Card to input analog signals into a computer as they are readily available and a standard in computing. Sound cards come in a wide range of models, typically professional use, gaming and consumer use.

Most modern computers have a sound card built into the motherboard, these often come with one or two line inn's and one stereo line out, and typically tend to be solutions from a company called Realtek. These are consumer quality sound cards and generally operate at a sampling frequency of 44.1 KHz.

Gaming and Audiophile solutions tend to be produced by Creative under the Sound Blaster range; they feature the same external interface that a motherboard sound card would, occasionally they will come with some extra line ins, but the main difference is that these sound cards operate at a higher sampling frequency, generally 96 KHz.

Professional grade sound cards tend to be produced by companies such as M-Audio, the main difference with professional sound cards is that they feature MIDI ports for the connection of music controllers such as mixers, synthesizers and keyboards. Typically these cards plug into PCI slots on the motherboard and can operate at a sampling frequency of up to 192 KHz.

Due to the popularity of laptops external sound cards that plug into USB are also available; these can be particularly cheap as a simple search on eBay can reveal hundreds of different models of USB sound card available for around 1 - 4 pounds, this includes postage. These typically come with one line in and one stereo line out, with a sampling frequency of up to 44.1 KHz

### **3.3 DAQ**

If a standard computer sound card does not meet required needs, for example a higher sampling frequency and bit quantization is required then Data Acquisition (DAQ) solutions may be the solution.

DAQ solutions are typically used in the medical industry, for example in functional magnetic resonance imaging, functional near-infrared imaging and X-ray computed tomography. Some of the main companies that produce DAQ solutions are National Instruments, Acquiretek, Omega, and Gage.

Sampling frequencies of 1 GHz and higher can be achieved, although the bit quantization tends to remain around 8 - 16bit although there are cards that offer 32bit quantization and higher.

DAQ solutions tend to come in a wide range of expansions for a computer,

USB, SATA, PCI-E, etc., whichever bus on a computer operates fast enough will generally be used, usually to the buyers discretion. DAQ expansions also tend to come with an on-board high resolution timer and C++ software SDK to interface with the hardware.

### **3.4 Audio Cable**

Audio cable is an important factor in audio solutions, if an audio setup doesn't have high quality audio cable when combined with a high quality sound card and microphone, the recorded audio quality will be lowered as it is subjected to the noise produced by the poor quality cable.

The main type of microphone cable is balanced & shielded audio cable, although any audiophile will argue about audio shielding. XLR connectors are most commonly used in conjunction with balanced cable. Balanced cable cancels out external interference such as noise.

When using audio cable with microphones the length of wire and it's resistance has to be taken into consideration; for example if one where to use a 10 meter length of audio cable with a standard piezoelectric microphone the signal would diminish due to the wire resistance before it got to the sound card, in this scenario a preamplifier would be required to amplify the signal so it has enough energy to get from one end of the 10 meter cable to the other.

## 4 SOFTWARE RESEARCH

### 4.1 Third Party Libraries

A software API is required to interface with hardware such as a sound card, generally operating systems provide you with a platform specific API to interface with standardised hardware such as sound cards however this limits the codes portability, when programming with portability in mind it may be best to use an open source LGPL library that is portable among a wide variety of platforms. It is important that it is an LGPL or similar license if you are designing commercial software because in a nut shell the LGPL licence allows you to use the code commercially for free and unlike the GPL licence it does not require you to release your source code to the public.

**Simple Direct-Media Layer (SDL)** is released under the LGPL licence, it is what it says in the name; a simple direct media layer. It exposes only a few simple functions for basic multimedia development that can be used as the foundations to build more complex functions.

It doesn't provide a set or get pixel function; rather the programmer is required to access the screen buffer pixels directly in memory, however SDL lacks luxury functions such as image transformations like rotation and scaling, these can be obtained in additional sub libraries such as `SDL_GFX` but the philosophy is that a function you write yourself is going to be better optimized for the specific use you intend to apply it to.

A useful feature of SDL is that it handles different types of image format behind the scenes, for example a bit block transfer of two different format images, a 16bit bitmap and a 24bit bitmap happens behind the scenes seamlessly. SDL does provide functions to convert image formats to increase performance. SDL also supplies an event messenger for input from joysticks, mouse and keyboard, some basic timing utilities and multi-threading functionality. SDL is supported on a wide variety of platforms, such as Windows, Mac and Linux.

**Port Audio** is not released under the LGPL licence but it is still released under a license that allows source code to be kept private and used commercially without paying a licence fee. Port Audio is a software library designed specifically for interfacing with sound card hardware; it too like SDL is supported under a wide range of platforms. Port Audio also has a relatively simple function set, allowing users to create call back functions that get triggered when audio data is dispatched from the sound card, essentially allowing a stream to be opened between the hardware and the software.

Using these two software libraries allows an application that features a Graphical User Interface (GUI) and access to sound card hardware to be written and ported to a wide variety of platforms.

SDL does supply an interface to audio hardware however it is only targeted towards writing audio to a line out and does not supply functionality for reading audio from a microphone.

## **4.2 Digital Signal Processing**

### **4.2.1 FIR & IIR Filters**

Infinite impulse response (IIR) Filters require previous input and previous filter output samples to operate, causing what is known as a feedback loop while Finite impulse response (FIR) Filters only require previous input samples and as such do not exhibit this recursive behaviour. The fundamental flaw with IIR filters is that because of its recursive nature the filter output can become unstable and oscillate infinitely.

IIR filters are more complicated to implement than FIR filters however they run a lot more efficiently, much more so than FIR filters because they often require fewer coefficients and this means less memory consumption and less multiplications. Filters are commonly used for low/high/band frequency passing, boosting and resonating. These can be particularly useful for filtering out undesired frequencies such as noise or interference.

### **4.2.2 Fourier Transform**

The Fourier transform is a part of Fourier analysis which expanded from the study of the Fourier series first introduced by Joseph Fourier.

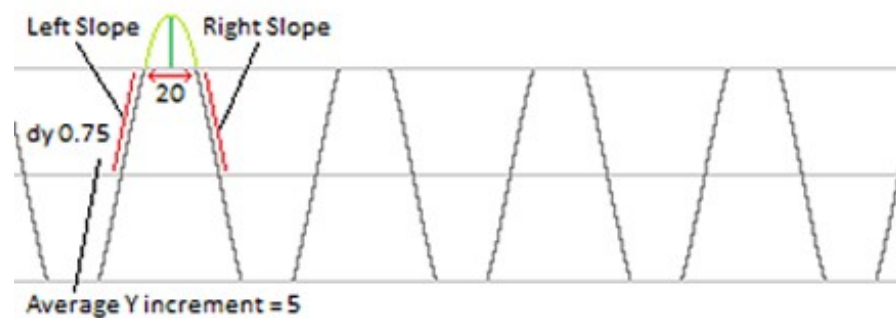
A Fourier Transform identifies the amplitudes of the individual frequencies a signal consists of; this operation essentially takes a signal from a time domain into a frequency domain.

In computing a Fast Fourier Transform (FFT) is most commonly used, an FFT computes the Discrete Fourier Transform (DFT) and its inverse. The FFT uses the Cooley–Tukey FFT algorithm, also known as the Danielson-Lanzcos Algorithm, which recursively splits a signal up into composite chunks in order to reduce computation complexity to  $O(n \log n)$ .

### 4.2.3 Intersample Interpolation

What is Intersample Interpolation? Well as I talked about in the chapter on Sound Cards a problem with quantization of an analog signal is often that the analog signal is under sampled and as a result clipping artefacts occur; this is called intersample clipping and it can be repaired using interpolation, referred to as intersample interpolation.

The following research into intersample interpolation was independently discovered using my knowledge of mathematics and digital signal processing.



As you can see from the diagram above intersample clipping has occurred, this is where the quantization of an analog signal doesn't have enough bits to quantize the entire signal, thus under sampling occurs and the ends of the signal get clipped, it is impossible to completely solve this and as I described in the hardware research a preamplifier or potentiometer can be used to scale the signal so that it is less likely to leave the quantization range but this does not completely solve the problem. A method of interpolation to approximate the height that the signal would have reached is required. This breaks down into three potential situations.

- If neither the left nor the right slopes are available, the best approximation that can be achieved is to take the width of the intersample clip (*in this case 20*) and to add that onto the clipped amplitude. This is shown on the diagram as the dark green line, as you can see this approximation can often be very accurate, although if multiple sinusoids are mixed down this method can become a lot less accurate.

- If both slopes are available the rate of change in the amplitude over time can be approximated for each of the slopes (*shown as  $dy 0.75$  on the diagram*) and used to continue the motion of the wave, this is shown as the light green line on the diagram and as you can see this is also a very accurate method of interpolating the maximum amplitude the signal may have reached.
- Only one of the slopes is available then the amplitude's rate of change over time can be calculated on just that slope.

Interpolating the signal using slopes is a time-consuming process when compared to just measuring the intersample clip length, so it would be sensible to detect if multiple sinusoids are mixed down into the signal and only use the more expensive interpolation if this condition returns true.

Detection if multiple sinusoids are mixed down into the signal can be achieved when calculating the amplitude's rate of change over time, if there is only one sinusoid then the rate should steadily decrease over time, if the rate were to suddenly jump higher at any point then this would be an indication that there are multiple sinusoids mixed down in this part of the signal.

An even more accurate result can be attained at the expense of more computational power and that would be to use the Fourier series to break the signal down into its approximate individual sinusoidal frequencies and to do an intersample approximation on each sinusoid, and from that compute the average intersample approximation.

## 5 MATERIAL RESEARCH

Understanding more about the common types of surface this technology will be used on is essential in understanding how the performance of the solution may vary when put into use.

### 5.1 Surface Material Properties

The speed at which sound travels through a material is defined by its elastic and density properties.

The more elastic a material is (*the less pliable a material is*) the faster a vibration can travel between molecules. This is because highly elastic molecules are able to return to a state of rest more quickly, allowing them to transfer vibrations from one molecule to another more frequently. A sound that travels through an object with low elastic properties (such as rubber) will travel a lot slower because it will take longer for molecules to return to their state of rest. As a vibration travels through elastic materials some of its energy is converted into heat energy due to the friction of vibrating molecules.

The denser a material the larger the molecules; it takes more energy to make larger molecules vibrate and as a result sound travels slower between denser molecules.

Sound will travel slower through materials with high elasticity and high density and faster through materials with low elasticity and low density. This can affect the accuracy of results calculated from discretely sampled signals obtained via microphones. Informing the software algorithm of what material the solution is being used on could help refine the accuracy of position detection and help determine a scale for calibration of mapping to screen space coordinates.

## 5.2 Material Surface Texture & Touching Stylus

Another factor that will affect the performance of the solution is the texture of the surface, surfaces with a smooth texture such as a glass window or whiteboard require very sensitive microphones to detect the vibrations made by an object being dragged across it, one method of increasing the sensitivity is to use an amplifier to amplify vibrations with a low amplitude so that they are detected within the range of quantization by the analog to digital converter, however if the vibration is too subtle for the microphone to pick up a more sensitive method of measuring vibrations will be required, for example a condenser microphone.



Smooth surface whiteboard ref ShopUK

Alternatively a stylus with a rubber end could be used, the friction of the rubber getting dragged across a smooth surface could cause the vibrations to be amplified just enough for the microphone to pick them up, and then an amplifier could be used to scale the amplitude into the range of quantization.



Rubber Tip Stylus ref GetPrice

Another method of amplifying the vibrations caused by dragging an object across the surface is to use a special type of stylus, for example a stylus which rolls across a surface using a rubber tipped cog shaped cylinder or

mine shaped ball. The impact of the extruded faces as the cylinder rolls across the surface would be similar to that of a little hammer; causing a more powerful vibration.

A good illustration is the pouncing wheel; although the edges on a pouncing wheel would need to be blunted and coated in rubber.



**Pouncing Wheel** ref DickBlick

Using a ball rather than cylinder would allow for more freedom of movement.

If a smooth surface is not absolutely vital then it would be better to use a slightly rougher surface such as wood, best results could be obtained from using the rough side of a plywood sheet.

### **5.3 Attachment of Contact Microphones**

Contact microphones can be attached to surfaces using duct tape, silicone or epoxy but the problem with these methods is that they are not very dynamic if it is required that the microphone is moved to new places often. The only reusable method available is to use Blu-Tack; the Blu-Tack must not be applied to the surface of the contact microphone but from behind the microphone, in the same way that a contact microphone would be taped to the surface. This reduces muffling of vibrations detected by the contact microphone.

The main problem with Blu-Tack is that over time it can lose its strength and contact microphones can fall loose, this is either due to the force of gravity pulling at the Blu-Tack or due to the force of a bendy wire pulling at the contact microphone.

## 6 ANALYSIS

### 6.1 Target Audience

The target audience can be generalised into four categories;

- **Tutors**
- **Artists**
- **Musicians**
- **Game Developers**

It is important that the requirements of each category are identified, thus some market research was performed which gathered the following results:

	<b>Tutors</b>	<b>Artists</b>	<b>Musicians</b>	<b>Game Developers</b>
<b>Product Questions</b>				
<b>Interested</b>	4/5	5/5	4/5	5/5
<b>Useful Rating (not) 0 – 5 (very)</b>	18/25	19/25	18/25	18/25
<b>Worth Rating</b>	£90 - £200+	£30 - £200+	£0 - £200	£30 - £200+
<b>Thought there where Advantages over existing solutions</b>	3/5	4/5	3/5	2/5
<b>Thought there where Disadvantages over existing solutions</b>	5/5	4/5	4/5	1/5
<b>Would Buy</b>	4/5	2/5	3/5	1/5

Although these results are not definitive due to the narrow volume of volunteers they do show some interesting information.

**A list of conclusions:**

- We can see that someone in each category was willing to pay up to £200 for the solution and in 3 out of 4 cases even more.
- In each category 4 out of 5 or more people were interested in the product.
- Most people thought the product was going to be useful.
- The product got an over average worth rating in all four categories.
- Most people involved in the survey saw advantages in this product over existing solutions.
- Most people involved in the survey also saw disadvantages in this product over existing solutions.
- Half of the people involved in the survey would buy the product.
- The results are fairly equal amongst each category.

**A list of conclusions from the feedback in advantages/disadvantages feedback: (Refer to Appendix G)****Quoted Pros:**

- “Would be good for drawing in 3D space.”
- “Has more variation.”
- “Allows for extendable canvas.”
- “You can use anything.”
- “Unlimited surface area.” x3
- “Large surface area.”
- “Multiple surface use.”
- “To wake up touch screen.”
- “If you could use anything as a touch screen graphics tablets would be a thing of the past”.
- “Seems very flexible compared to the likes of touch screens.”
- “Could replace traditional peripherals allowing you to use anything as a controller and would be interesting as a new method of controlling a game.”

- “I can see it being good for education.”
- “Adds a new alternative and a new control surface. New way to interact with instruments and software.”
- “The real thing is always better.”
- “More portable and interactive.”
- “Cheap, unique.”

As you can see the pro’s outweigh the con’s which is a good start.

#### **Quoted Cons:**

- “In reality it would probably be too expensive.”
- “Not specialised.”
- “Sounds like the Wacom Cintiq.”
- “Possible loss of accuracy?”
- “I would only find a monitor useful for use as a touch screen.”
- “Too much of a faff on. Expensive?”
- “May not integrate with current packages.”
- “IPAD 2 has this technology and more.”

#### **Here is a break down of the feedback:**

- I did not mention the price which the product may be available for and thus there were some concerns of an expensive price tag, this product has been designed to be inexpensive and can be produced for around £30 using off the shelf components, even less if put into mass production.
- There are concerns that the setup of the project may be cumbersome, if using laser microphones the setup is relatively easy and simple (*refer to recommendations, laser microphones*)
- The IPAD 2 voids the whole point of this project’s particular attributes.
- It is reasonable to assume a loss in accuracy, this is possible however a range of different models could be put into production to settle the accuracy needs of different consumer uses.

- It is true that because this project is not a specialised solution it may not fair as well as other solutions would that have been designed specifically for the problem at hand.
- It is possible that it may take some time for current mainstream packages to adopt specialised interaction with the surface however the basic functionality will work out of the box with most graphics applications, it is musicians and gamers that may find compatibility more of a problem.

## 6.2 Computer Vision Approaches

Currently the commercial standard for creating touch surface and touch screen projectors uses computer vision techniques such as object tracking.

Computer Vision Object Tracking techniques rely on producing depth or disparity maps from 2D webcam images by calculating the disparity between each pixel. There are a range of methods to produce this, more advanced systems like the Microsoft Kinect shine an infrared light and then use an infrared camera to take a photo of the infrared light, when this is performed in indoor environments it excludes a lot of other unrequired information making the calculations faster and more accurate however in an outdoor environment the infrared light from the sun can be a large source of interference.

The most popular method of calculating a disparity map for real-time object tracking is the Sum of Squared Differences (SSD) method, although there are a wide range of other types of map that can be produced.

Even using the popular Sum of Squared Differences method to track objects in computer vision the computational cost of each frame is phenomenal, not only are methods like Sum of Squared Differences high in complexity requiring numerous multiplications but even more so on the amount of data they have to process.

On average a computer vision system has to process at least 25 frames per second to be responsive enough for human use. Even though modern computers can handle this kind of computational cost, too much computational power is being used for tracking input and very little is left for running computer applications.

There are also problems in the scalability of computer vision touch surface technology, in order to improve accuracy of object tracking the camera needs to take larger resolution images which takes up more buss bandwidth and memory, which is known to be one of the main bottlenecks in computer

programs; also it will take a lot longer to compute larger frames from the camera because the per pixel information increases exponentially as computer images get scaled up in size. It is a vice-versa situation, if you want to increase the speed of your application you will have to lower the resolution of the images you receive from the webcam and thus you look at losing an exponential amount of tracking precision.

The only math involved in the proposed technology is a 2D trilateration algorithm which is at the expense of just a few multiplications and divisions, complexity of  $O(1)$ , and for the case of multi touch detection you may want to use a Fast Fourier Transform (FFT) at the complexity of  $O(n \log n)$ .

Even with multi-touch detection using an FFT the proposed solution works out much faster than a computer vision solution; for a start this is partly due to the smaller amount of data that needs to be processed to achieve the same precision.

To put this into perspective a sample of 16bit audio is accurate to 65,536 values for position, at the cost of 2 bytes. To achieve this using a computer vision solution using a webcam you would need to take each image at a resolution of 256x256 which is a cost of 196,608 bytes (192 kilobytes) per image. That's 98,304 times more expensive in memory and USB bandwidth.

The proposed solution uses four microphones so the cost is 8 bytes for accuracy up to 262,144 positions per sample.

Computer vision equivalent would be 512x512, 786,432 bytes (768 kilobytes) for accuracy up to 262,144 positions per sample (image).

The proposed solution can process 44100 samples every second of 8 bytes each accurate to 262,144 positions, a total of 352,800 bytes (352 kilobytes) per second.

A computer vision solution on average is processing 25 samples per second of 786,432 bytes each accurate to 262,144 positions, a total of 19,660,800 (18.75 megabytes) bytes per second.

To put this into perspective;

	<b>Computer Vision using Object Tracking &amp; Webcams</b>	<b>Proposed Solution using Digital Signal Processing &amp; Microphones</b>
Frequency of Buss Transfer <i>(how responsive)</i>	25 HZ (Slow Response)	44100 HZ (Fast Response)
Memory Bandwidth of Buss Transfer	<i>512 x 512 Image</i> <b>786,432 Bytes</b> ~200,000 Bytes if JPEG Compression.	<i>4x 16bit Buffers</i> <b>8 Bytes</b>
Total Memory to Process Logic on every Second	19,660,800 Bytes	352,800 bytes

As you can see the proposed solution works out to be 1764 times more responsive than a computer vision solution at 55.7 times less expense on memory and bandwidth, which will proportionally affect the computational expense too.

The proposed solution also uses less than half the memory every second than the computer vision solution uses every buss transfer!

Images from a webcam are usually not in raw format, typically compressed using JPEG compression, which comes at a trade-off less memory consumption but at the expense of more computation to decompress the images once they are received. JPEG decompression is at the expense of a

2D Discrete Cosine Transform which is a complexity of  $O(n \log n)$  and can be more expensive than an FFT.

The big advantage that a computer vision based system has over the proposed solution is that the computer vision solution will work with any surface as mine will only work with surfaces that sufficiently carry surface vibrations and thus brick walls are not compatible with this project.

### 6.3 Performance Analysis

While developing the solution I independently discovered and developed a trade-off philosophy regarding overall performance of the solution which can be summarised as the responsive, accuracy and speed trade-off.

The problem is best defined as a table that uses ratings of 0 – 10, 0 for low/slow and 10 for high/fast. The greyed out numbers are so because they are irrelevant/obvious statements.

	<b>Feedback Response Time</b>	<b>Results Accuracy</b>	<b>Logic Speed</b>
<b>Responsive Feedback</b>	10	5	10
<b>Accurate Approximations</b>	5	10	5
<b>Fast Logic</b>	10	5	10

From the results in the table, we can make three statements;

- If you want responsive feedback you're going to have to give up some accuracy in the results which means making the logic code simpler and faster.

- If you want accurate approximations you're going to have to give up some feedback response time, and give up some logic speed by making the logic code more complicated.
- If you want fast logic, you need to make the logic code simpler at the expense of losing some accuracy in your results.

This really boils down to two configurations from a consumer perspective;

- Responsive Feedback over Accurate Approximations  
*or*
- Accurate Approximations over Responsive Feedback

Fast Logic directly affects the timing of responsive feedback, thus it is removed from the equation.

But the question is; what is the better trade off? Well that is a question that in this economic climate boils down to what hardware is at disposal, which boils down to how much money the consumer has to spend on the product.

It is not a matter of preference, under these rules the same hardware cannot have different balances or configurations of this trade-off philosophy. There is one satisfactory/optimal balance for each hardware configuration. If precision is a preference, hardware needs to be developed that can sustain a suitable balance of this model which favours well in Accurate Approximations.

One example is; from this model it is plain to see that if Accurate Approximations are desired the speed of the logic code will decrease and as a result the feedback response time will take longer. Nothing will change this, this will always be a problem, but if implemented on hardware that can process logic fast enough for the feedback response time to become unnoticeable or 'satisfactory' then we have the optimal trade-off balance. Otherwise, the optimal trade off balance may have to be in the favour of less accurate approximations.

## 6.4 Multi-Touch Detection

Multi Touch Detection can be best achieved by using a set of Unique electronic Stylus, which inform the software, e.g. via infrared or Bluetooth, which of the Stylus is currently being pressed against the subject surface.

However, if it is required that any type of stylus can be used to interact with the surface such as human fingers then only touches between objects which vibrate a noticeably different set of frequencies when touched or dragged along the surface can be distinguished. For example a pen and a pencil may vibrate significantly different frequencies because a pen has a metal ball tip and a pencil has a malleable graphite tip. This may lead to the development of a universal stylus which has an interchangeable tip which when dragged across a surface will vibrate unique frequencies, the advantage to this is that no electronics would need to be used in the stylus reducing cost and the annoyance of replacing batteries. Also, there could be a wide range of specific tips for different textures of surface.

This breaks down to three solutions to Multi-Touch Detection;

- **Method One;** Unique Electronic Stylus which has a pressure sensitive tip which sends a remote message to the software informing it of its unique stylus ID and that it has been pressed against the surface.
- **Method Two;** Detection of different objects from the unique frequencies resonated or existing in the signal produced by the touch or drag of the chosen stylus object.
- **Method Three;** A universal stylus with interchangeable tips which are designed to resonate or produce unique frequencies when tapped against or dragged along a surface.

While none of these are a definitive solution, rather they are all equally as valuable solutions as each other, but should be applied depending on the scenario.

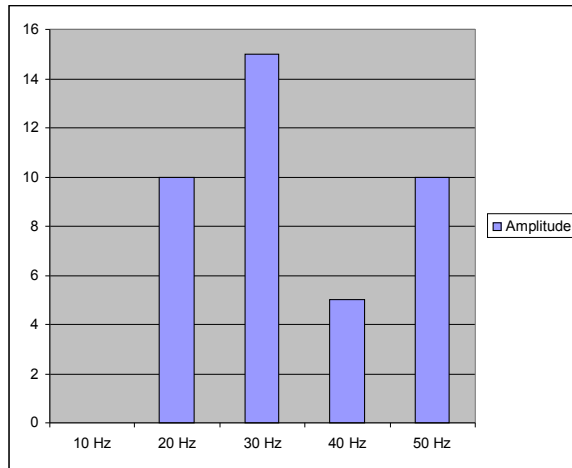
A great feature of Method Two and Method Three is that they both work hand-

in-hand if one of them can be achieved, then the other can be achieved without any extra technology or modifications. So people could purchase a fully dynamic solution with multi-touch detection without the requirement of a unique stylus and if they wish for better precision they can buy the universal stylus and some unique heads. A good example of a customer that would benefit from this is an artist as they often require a wide range of different texture and shape stylus mediums in traditional art to achieve different type brush strokes.

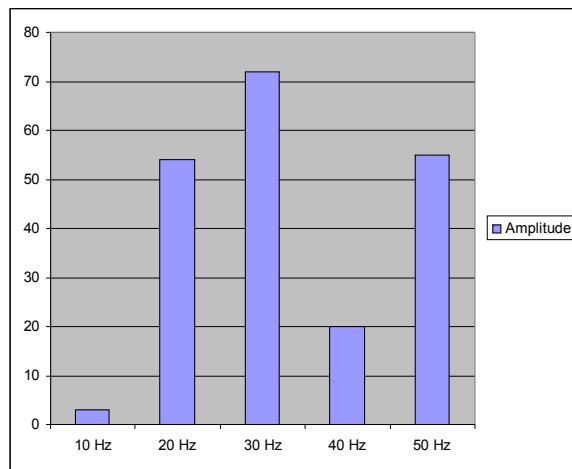
The analysis of frequencies produced by a stylus dragging or touching a surface a Fast Fourier Transform would be required to transform the signal from a time domain to a frequency domain. We can then identify what frequencies are present and the amplitudes of each present frequency. Objects intend for use on a touch surface will need to go through an initial calibration process before they can be used, this will consist of dragging the object across the surface and tapping it against the surface two or three times so that that a signature of the frequencies it produces in both scenarios can be built.

How this works is that each time a stylus is dragged across the surface, some frequencies will be present and some wont, it's the frequencies that are always present in each drag and have high amplitudes that we are interested in, frequencies with high amplitudes are referred to as resonant frequencies. Once we have detected the core frequencies that are produced by the object dragging along the surface we need to work out the approximate amplitude range those frequencies differ from each other, this is because depending how hard a surface is touched the amplitudes will be different in scale, however the difference in scale between each amplitude should always be the similar within a range of tolerance, and that is what we are looking to detect, the range of tolerance between each frequency amplitude.

I have produced two charts as a simplified example to help illustrate this;

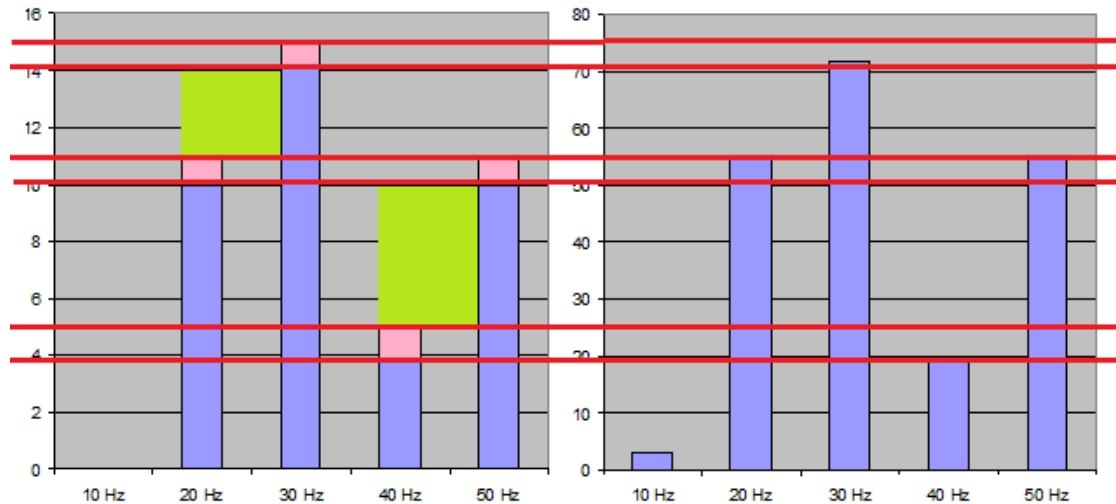


In this chart the surface was touched with some force, the results show that four frequencies were present, 20, 30, 40 and 50 Hz. All of the amplitudes here are of a maximum scale of 16.



In this chart the surface was touched with exactly five times the amount of force as last time. This time we see that all frequencies are present and that the frequencies present in the last chart have roughly the same difference in size between each other but overall the individual amplitudes are at a much higher scale of 80. The 10 Hz frequency although having a small amplitude, because it is so small it is considered an inconsistency or artefact of noise or

interference and ignored.



In this comparison image I have highlighted the fixed amplitude differences in green and the amplitude difference tolerance in pink to illustrate the pattern between the two that can be used as an identification signature of that stylus/object.

## 6.5 Double Buffering

Any computer program that deals with audio needs a method of buffering, this is because there is a good chance that the computer will not be able to process the information being received fast enough to keep up with the rate at which new information arrives, and as a result a lot of new information is lost. Traditionally double or triple buffering is used which is satisfactory if data is just being stored in memory, which is what an audio recorder application is, however if a computationally expensive / time consuming algorithm is being performed on the data in real-time then higher orders of buffering will be required.

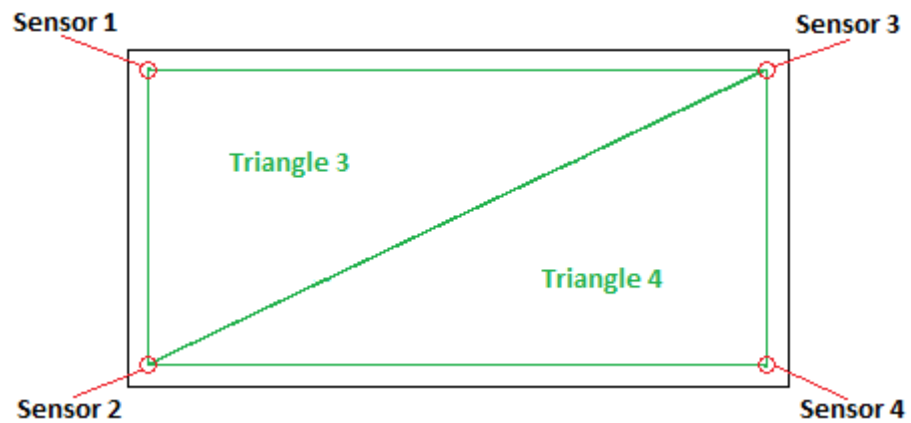
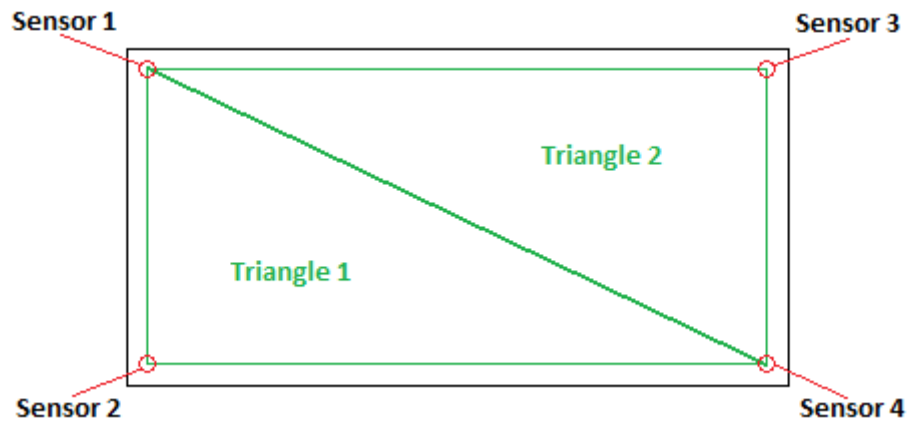
What an audio buffering system does is store all of the received data in an array so data in the array can be worked through at a slower pace than it is

being received. A problem with this is that over time the latency between received data and the data that you are currently processing logic on will increase until it becomes evident to the user. To resolve this, a maximum latency needs to be maintained, if the latency increases over the maximum latency then some of the received data in the array buffer needs to be discarded.

## **6.6 Trilateration**

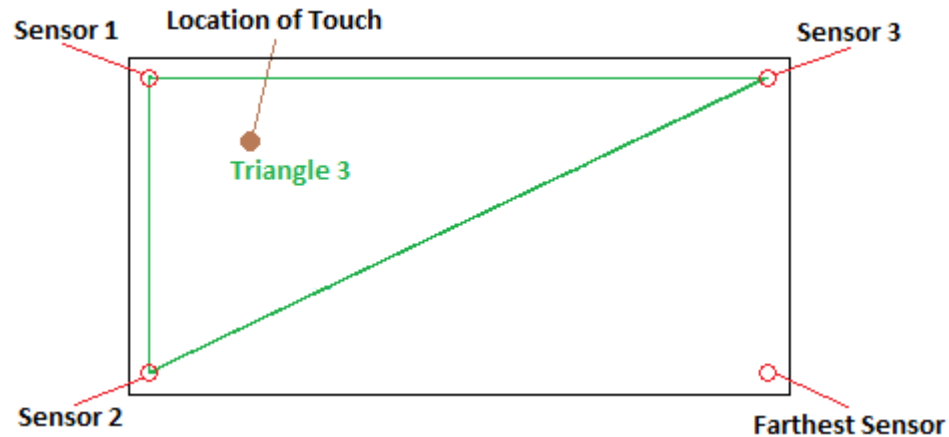
Trilateration requires six inputs to calculate a 2D Cartesian coordinate output. The first three inputs are the positions of the three microphones that surround the touch position in a triangle shape and the last three are the delays to the position the surface was touched. Unfortunately in this solution trilateration is not as straight forward and needs a bit of re-working.

There are four microphones, these make a total of four triangles as illustrated below;



The amplitude information received at each microphone can be used to calculate the position where a touch occurred.

To do this we must first identify which triangle the touch is located in, this is a simple process of working out which sensor is the farthest away as illustrated below;



We can plainly see that the location of touch is within Triangle 3, but the algorithm detects that the farthest sensor is Sensor 4 and thus the location of touch must sit in a triangle formed by Sensor 1, 2 and 3. The rule is simple, exclude the farthest sensor from the location of touch and the remainder three sensors are the ones required to identify which triangle the touch origin is located in. The farthest sensor is identified as being the sensor to receive the lowest amplitude. As a vibration travels farther its amplitude diminishes.

Once we have identified the three sensors that form the triangle around the origin of the touch, we can work out the Cartesian coordinate of the touch origin by using the amplitude received by the three identified triangle sensors as the delays for the trilateration algorithm.

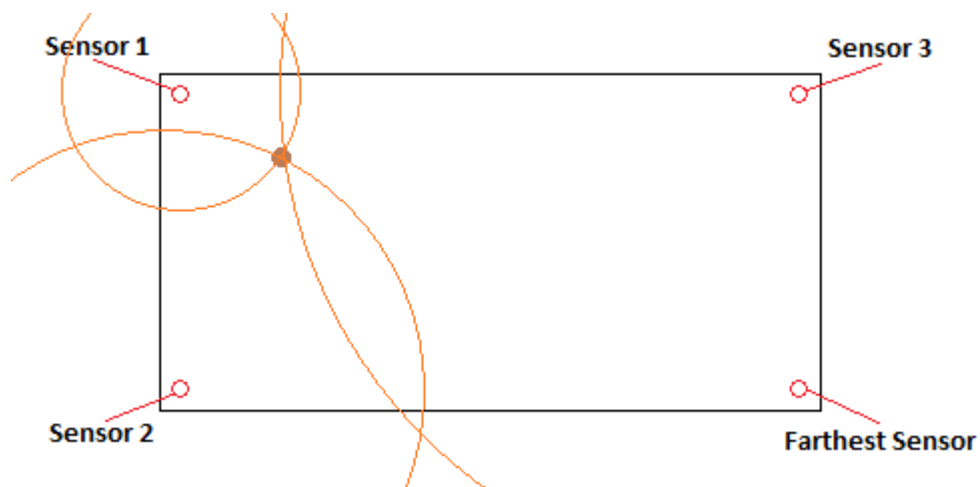
However first the amplitudes need to go through some formatting; the magnitude component needs to be removed from the amplitudes so that the output coordinates are not affected by the different forces a user can touch the surface with; this is achieved by calculating the magnitude of the three amplitudes and dividing each amplitude by the magnitude, the resulting three amplitudes are now scaled to a 0 – 1 range.

Secondly the amplitudes need to be inverted, this is because the further a vibration travels the smaller it's amplitude becomes; as you will see below when put into the trilateration algorithm unless the delays are made to represent the distance travelled there will be no intersection of bounding radius, in short the further a vibration travels it should be represented by a

value that gets larger rather than one that gets smaller. Because the amplitudes are now in a 0 – 1 range after the normalisation the inversion can be easily achieved by individually subtracting each of the three amplitudes from 1 e.g.  $(1.0 - 0.75 = 0.25)$

The trilateration algorithm calculates the Cartesian coordinate of the touch origin by working out the intersection point of three circles around the sensor locations, all with a radius relative to the delay set by the sensors amplitude.

The diagram below shows this process in more detail;



As you can see, the circles made from the inverted amplitudes received at each sensor indicate the touch origin from the point at which all three circles intersect.

## 7 DESIGN

### 7.1 Hardware

The Hardware design consists of four cheap USB sound cards that operate at 44.1 KHz, an externally powered USB hub to ensure that all four sound cards get enough power, four balanced audio cables at 3+ meters long and four piezoelectric contact microphones and a plastic chopping board with a crosshatch textured surface of bumps and four rubber stoppers to act as little legs, over time however the plastic chopping board was warped by gravity and has bent in the centre due to the lack of rubber supports in that area. The chopping board is used as a test surface environment; the surface is generally touched with a variety of pens that litter the immediate area.

### 7.2 Pipeline

The pipeline is composed of four main functional stages; the first two being input preparation stages and the last two being data analysis stages:

- **Microphone Input Handler**

This module will be responsible for the configuration of audio devices and will provide a simplified interface for access to audio hardware.

- **Binary Double Buffer**

This module will be where the data gets cached while it waits to be processed by the main analysis parts of the pipeline.

- **Touch Detector**

This is the first Analysis stage where data is analysed on a per microphone basis for possible touch signals, it will determine if a touch has occurred, how long for the touch occurred and force at which the touch occurred.

- **Trilaterator**

This is the second Analysis stage where data is processed as a group, this stage only becomes active if all four microphones passed the touch signal detection in the Touch Detector stage; this stage determines an approximate time delay between each microphone using one of three

approximations (sample counting *function of binary double buffer*, please refer to 7.3, high performance timer or an average of the both) but more importantly it will approximate the position at which a touch vibration originated by normalising and inverting the amplitudes determined by the Touch Detector stage for each microphone and using the transformed amplitudes as the delay data for the trilateration algorithm.

Besides these four main stages there are two smaller stages which allow the user to interject the pipeline with configuration data, which are described below:

- **Calibrator**

This stage is directly and only accessible to the Trilaterator; it is where the configuration information for the origin of each microphone is stored.

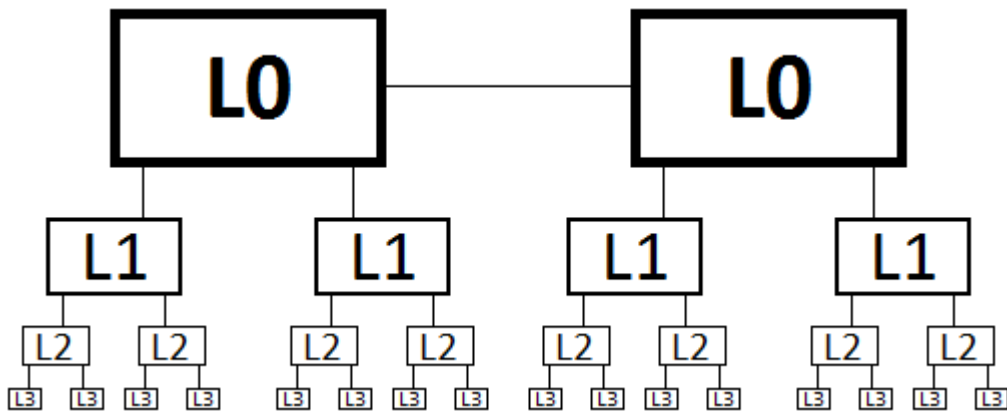
- **Detection Thread**

This is the glue that sits between the Binary Double Buffer, Touch Detector and Trilaterator it performs as a method of user input configuration between the Touch Detector and Trilateration analysis stages as well as being a fundamental part in connecting the stream from the input data caching up in the binary double buffer to the analysis stages of the pipeline with low latency response time.

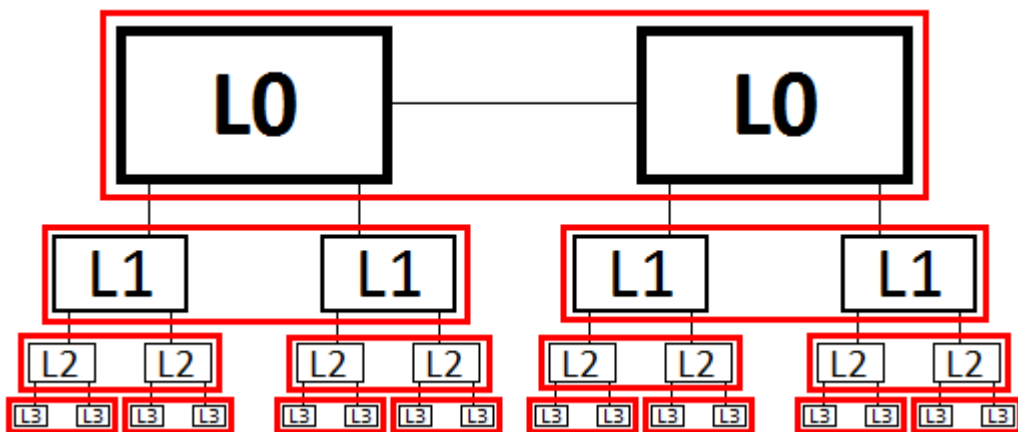
Please refer to Appendix B for an overview diagram that illustrates the pipeline and the relationships between its stages.

### **7.3 Binary Double Buffer**

The binary double buffer illustrated below is a multi-buffering technique that I independently developed simplifies to a binary tree of double buffers.

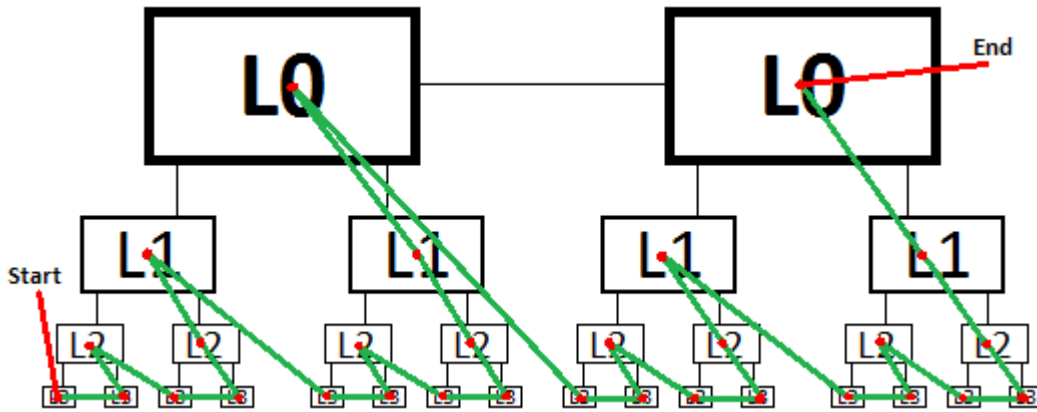


As you can see, at level zero is a standard double buffer. Each fork is just a standard double buffer; here is an illustration which highlights each individual double buffer;



Here you can see each double buffer has been circled in red; there are a total of 15 double buffers in this solution. This was found to be the optimum number of double buffers to cache 16bit data coming in at 44.1 KHz for processing by the touch detection pipeline, the results for this can be found in the Profiling & Testing chapter.

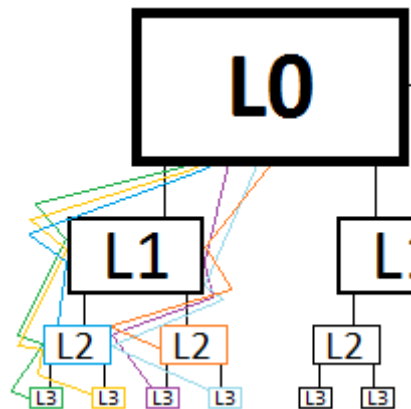
Blow is an illustration showing the write access path of transversal through the binary tree, every time a new chunk is added to the tree it is added according to this linear sequence, this helps maximize cache efficiency and allows for optimizations such as hard coding branches allowing the compiler to better optimize branching and avoid wasted operations for looping over arrays.



The linear path is illustrated in green and the nodes visited along the path are illustrated as red dots.

As you can see write access starts from the bottom left and works its way up to the top right.

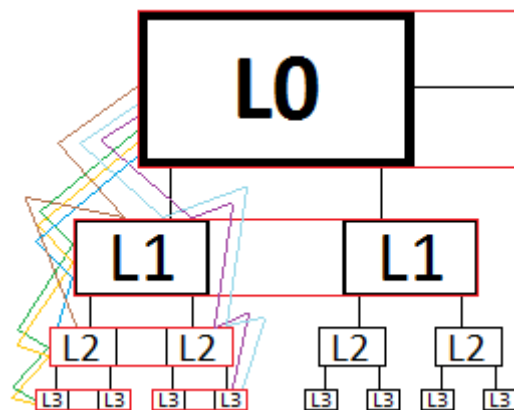
This path is a product of how data is added to the tree, when the `AddChunk(float*)` member of the Binary Double Buffer is called it finds the deepest left aligned node that is empty to store the chunk in, the diagram below demonstrates how this works with six `AddChunk()` calls illustrated in different colours.



The first AddChunk() call (green) checks if L0 is empty, if it is it then checks if L1 is empty, if it is, it then it checks if L2 is empty, and if it is, it checks if L3 is empty, yes it is – now because it has reached a leaf node the data is stored here and aligned to the left of the L3 double buffer.

The second AddChunk() call (yellow) checks if L0 is empty, if it is it then checks if L1 is empty, if it is, it then it checks if L2 is empty, and if it is, it checks if L3 is empty, there is space for more data on the right side of the L3 double buffer and thus the data is stored there.

If this doesn't click straight away then the illustration below simplifies the process by looking at pairs of nodes as double buffers.



The red rectangles outline the pairs that make double buffers.

The first AddChunk() call (green) checks if L0 has free space, it does, so it checks L1 and that does too, so it checks L2, that does too, so it checks L3 that does too but because this double buffer is a leaf node, the data gets stored there.

The second AddChunk() call (yellow) checks if L0 has free space, it does, so it checks L1 and that does too, so it checks L2, that does too, so it checks L3 that does too but because this double buffer is a leaf node, the data gets stored there.

The Third AddChunk() call checks if L0 has free space, it does, so it checks L1 and that does too, so it checks L2, that does too, so it checks L3 and that doesn't so it stores the data in the L2 double buffer.

Getting data from the binary double buffer has to be done in the same linear order that data is added otherwise the order of the samples will be lost and in audio it goes without saying that the order that the samples are in is very important unless you're an electronic musician experimenting with granular synthesis.

A counting system has been added to the binary double buffer so that timing can be obtained from counting the amount of samples received. For example if it is known that you are receiving samples at a fixed rate of 44,100 times a second you know that when you have received 44,100 samples a second has passed. This is an alternative to using high performance counters which can come at a higher cost on performance, although be aware that the resolution is limited to the resolution of your sample rate.

While the system may seem complex to explain, the code is minimal and inexpensive, the sheer size of code is due to the expansion of if statements from iterative loops to maximize performance using a technique known as hard coding, please refer to Appendix D.

The performance of this module has been analysed in the Profiling and Testing chapter where an analytical break down of its advantages and disadvantages are available.

#### **7.4 Touch Detector**

This Stage of the pipeline is specialised to deal with detection of touch vibrations and dragging gestures and takes three user configurable inputs;

- **Touch Tolerance** (*Scalar*)

A touch is detected by monitoring the amplitude of the time domain

signal, when the amplitude is detected to be over the tolerance amplitude level then a touch is registered.

- **Touch End Wait** (*Time in milliseconds*)

When a touch is initially detected, the touch detector will allow a time period where no amplitudes over the tolerance amplitude level are received and still consider the user to be touching the surface, this is useful for when detecting gestures such as dragging because occasionally the amplitude of the vibration caused by a dragging operation can drop below the amplitude tolerance level for 5 – 15 milliseconds at a time.

- **Fix to Highest** (*Boolean*)

This is an option which when enabled stores the highest received amplitude as the touch force, if this is not enabled then the force from the last detected amplitude over the tolerance amplitude level is used.

The output from this stage is the amplitude force at which a user touched the surface with and how long the touch occurred for, a prolonged touch for example would be considered to be a dragging gesture.

A hidden function of the Touch Detector is that it is responsible for noise cancellation; each microphone device has its own touch detector assigned to it so the noise cancellation is on a per microphone basis, the way this works is that for the first n amount of received data chunks from the Binary Double Buffer where there is no user input the amplitude range is averaged and stored as the noise cancellation amplitude range, meaning that is the range of amplitudes that are ignored.

Another hidden function of the Touch Detector is that this is where intersample interpolation is performed, when detecting the touch force the intersample interpolation function is used in the case that intersample clipping occurs.

## 7.5 Trilaterator

This Stage of the pipeline approximates positions using the trilateration algorithm from the amplitudes approximated by the Touch Detector; the stage has three user configurable inputs;

- **Timing Type**

although the timing data is not required or used by any part of the pipeline it is a useful and fundamental debugging statistic to see which hardware is responding faster. There are three different timing approximations, the first uses a high performance timer in this case the performance counter of one CPU on your computer, the second uses the sample counter provided by the binary double buffer and finally the last is an average of the two timing methods.

- **Touch Timeout** (*Time in milliseconds*)

This defines the maximum time period the Trilaterator will wait between receiving the first touch detection signal from a Touch Detector and receiving the last, for example if the touch time out is set to 100ms and it takes 150ms for all four touch detectors to detect a touch signal then it will not be considered a touch and as a result an approximate position will not be calculated because the detections didn't occur in a satisfactory time period to be directly related to one another which in this scenario was 100ms.

- **Next Calculation Delay / Wait** (*Time in milliseconds*)

If all touch detectors detect a touch signal within the Touch Timeout period and an approximate position is calculated this variable defines how long the Trilaterator will wait before producing another approximation, this helps increase performance and reduce computational cost for dragging gestures where a high resolution of tracking is not a requirement, or the multiple triggering of position calculations from the same touch vibration.

### **7.5.1 Calibrator**

This is one of the simpler modules, each microphone is represented by an integer index, and this is where the integer index gets referred to a normalized screen space coordinate configurable by the user; a glorified lookup table in all respects.

### **7.6 User Interface**

The user interface provides a technical overview of the running solution allowing for visual debugging and pipeline configuration. The output of each microphone is displayed in four individual oscilloscopes; the oscilloscopes have two red and green horizontal coloured lines across them, the red one represents the noise cancellation amplitude range and the other one represents the touch tolerance level.

The touch tolerance level can be adjusted using the threshold sliders, these are represented as LED's which reduce and increase in brightness.

The coordinate/calibration information for each sensor can be adjusted, as well as all of the pipeline inputs.

Finally there is an output pane where the output statistics from the last touch are displayed, the touch start and end times will slowly turn from green to red if a new touch is not detected for some time, this is to notify the user that the currently displayed information is expiring as it represents a touch that happened quite some time ago and does not represent a recent event.

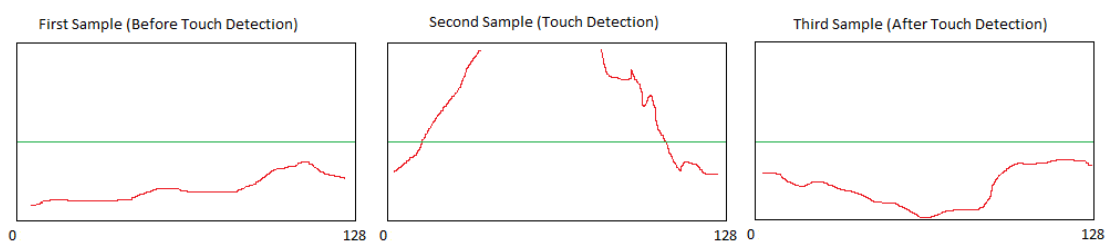
The user interface is designed from the ground up and only uses SDL to draw to the screen; each control item has been designed from scratch specifically for the scope of this application.

## 8 IMPLEMENTATION

### 8.1 Intersample Interpolation

The intersample interpolation proved to be one of the more troublesome functions to implement and was a vital part of correctly detecting touch amplitudes.

From reading about Intersample Interpolation in the research chapter we know that for best results we require both the left and right slopes leading up to the intersample cut off, so it is the job of the touch detector to ensure this algorithm receives a sample which contains this information. This is achieved by stitching three samples into one large sample; the first part is the last sample before a touch was detected, the second part is the sample where a touch was detected and the last part is the sample after the touch was detected. So in theory with those three samples, we should have something that looks like this;



although a crude illustration you can see how the three samples can capture the entirety of the slopes leading to the touch, the green line defines the amplitude range that needs to be exceeded before a touch is detected. Because the audio buffering system buffers in chunks of 128 floating point values I have illustrated the scales from 0 to 128.

Once the algorithm has a suitable sample to analyse the implementation breaks down into nine steps;

- Loop through the array and log amplitudes that are over 0.90.

- If there's a sequence of amplitudes over 0.90 then log that as an intersample clip in the clipping array.
- If there is not a sequence, log it as the highest detected force.
- Once the loop has completed, if the clipping array is empty, return the highest recorded amplitude.
- If the clipping array is not empty, loop through it and identify the largest intersample clip.
- Detect the start of the left slope and detect the end of the right slope.
- Work out the average increment in amplitude of the detected slopes.
- Work out the average increment reduction for the detected slopes. (how much each average increment reduces in size over time)
- Calculate results using detected slopes; otherwise just return the intersample clip length multiplied by two.

The performance of this algorithm can be improved by the ignorance of the negative spectrum, which is that the algorithm above describes. It works under the assumption that if the amplitude peaks a particular height in the positive spectrum it almost always peaks approximately the same in the negative spectrum. Checking both spectrums individually would double the computational complexity of this algorithm, the accuracy attained by doubling the complexity of the algorithm is most probably not worth the trade-off.

The source code revisions of this function are available in Appendix E.

## **8.2 Prototype Implementation**

The prototype implementation of this technology for demonstration purposes was a piece of plywood which acts as four musical percussion pads, all of the audio was programmed to be generated in real-time and thus the user interface had a Synthesizer Configuration menu added to it so that the sounds produced by each percussion pad could be changed, also a Korg Kaossilator sat between the sound card and the speakers to add additional delay effects, this helped demonstrate the full potential of this technology as a musical instrument. Although this technology was initially designed more for artists,

the precision in touch position detection was too poor for it to be impressive.

## 9 PROFILING & TESTING

### 9.1 Binary Double Buffer Performance Analysis

An analysis was performed to determine two questions about the Binary Double Buffer;

- **What kind of performance is the Binary Double Buffer running at?**
- **Are chunks at the end of the buffer getting left and pulled out after extended periods of time in the wrong sequence to which they were originally input? The tree somewhat resembles stack like access, which can cause serious problems, but this needs to be proved / disproved.**

Below two tests have been performed on the binary double buffer output to gauge the performance.

In the following analysis I refer to Chunks as blocks of 128 floating point samples.

#### **An analysis in debug mode shows the following:**

Lost Chunks: 581

Waiting for Chunks: 81370

Recorded Chunks: 44699

Processed Chunks: 44685

That works out to 1,686 lost milliseconds of audio out of 129,738 milliseconds. Put into perspective that's a millisecond lost every 76.95 milliseconds, approximately 1.3% every minute.

#### **Workings:**

$44100 / 1000 = 44.1$

*(44.1 samples = 1 millisecond of audio when sampling at 44100 Hz)*

$581 * 128 = 74368$  (*Lost Chunks converted to lost Samples*)

$74368 / 44.1 = 1686.35$  (*Samples converted to Milliseconds*)

**Another analysis of a very similar time period in release mode shows similar results:**

Lost Chunks: 554

Waiting for Chunks: 86768

Recorded Chunks: 45974

Processed Chunks: 45974

That works out to 1,607 lost milliseconds of audio out of 133,439 milliseconds. Put into perspective that's a millisecond lost every 83 milliseconds, approximately 1.2% every minute.

**The two results show that only a small amount of data is being lost,** 1.2% loss of data every minute is an acceptable loss of audio in this particular real-time application, intuitively it would not be worth the computational expense of adding another level to the binary double buffer to resolve a 1.2% loss every minute. The results however also show that the Detection Thread (main loop) is attempting to access the buffer for data faster than the buffer can keep up with, in-fact the results show that it's going three times faster than it needs to be. Currently the thread loops every millisecond; the results show here that the thread loop could be increased safely to approximately 3 milliseconds although such an optimization is not at all necessary the performance gained over the accuracy that could be lost is negligible.

**A test was carried out where each Chunk in the double buffer was assigned a timestamp** for when they were added to the tree, each time a chunk was taken out of the tree the timestamp would be compared with the current time and the highest 'expiry time' ( $abs(time - timestamp)$ ) was logged. The results showed that often the highest expiry could be nearly as large as a second, this is conclusive proof that samples are being left at the bottom of the bucket so to speak and forgotten about because the latency of the solution is less than one or two milliseconds thus no chunk should have an

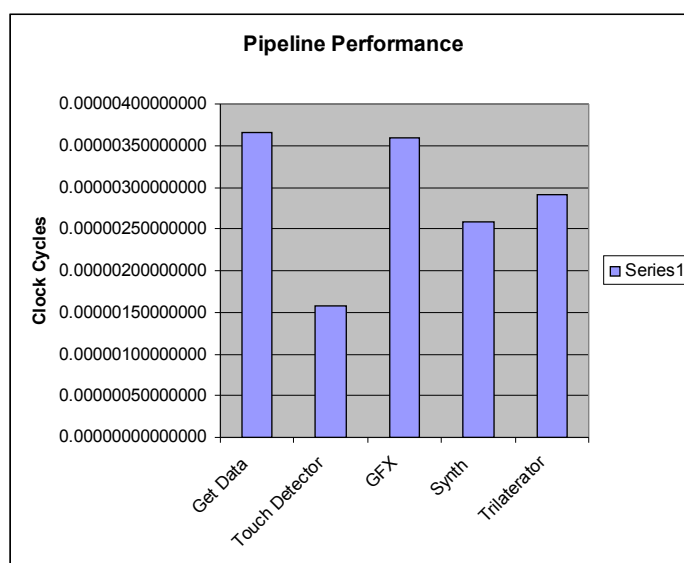
expiry time larger than this, this is also compelling proof that the binary double buffer will in-fact be disordering all of the audio samples and thus directly affecting the accuracy of the entire application.

The solution carried out was to replace the buffering system with a STL vector container of Chunks (128 length floating point array) and perform Add / Get operations in a queue type manner.

## 9.2 Identification of Pipeline Bottlenecks

Below is a table which represents the average performance of the Touch Detector Pipeline (TDP) over one minute, this data was performed using a standard buffering system and not the Binary Double Buffer. The TDC thread is responsible for the following operations:

- Gets a 1x chunk from 4x audio buffering systems (for each mic).
- Writes each chunk to a GFX Oscilloscope.
- Checks each chunk to see if a touch occurred.
- Ticks the Trilaterator to calculate positions.
- Push backs Mixer Samples into the Audio Mixer when sounds are triggered.



The data shown in the graph above is surprising because shows the modules which have the more complicated algorithms such as the Touch Detector and Trilaterator take the least time; in the Touch Detector this is because the expensive intersample interpolation is only performed when an amplitude

larger than 0.9 is detected and even then not the entire algorithm gets utilized unless more values of 0.9 are detected in a row, thus the algorithm is used sparingly, and the same goes for the Trilaterator, the most expensive code is only executed when all four touch detectors register a touch.

Getting Data from the audio caching buffers and updating the GFX Oscilloscopes are the most expensive because the entirety of the code is performed on every tick, this means that every time the TDC loops four 512 byte floating point buffers are being copied from memory to be analysed by the Touch Detectors. I could have optimized this by accessing the data directly from the audio buffer before removing it from the buffer; rather it is currently being copied into the memory space of the TDC and then removed from the audio buffer. A similar operation is occurring for the GFX Oscilloscopes, although for a practical application there wouldn't be a need for GFX Oscilloscopes unless in debug mode so this is not a significant problem.

The bottle necks in the TDC are evidentially Retrieving Data from the Audio Buffers and writing debugging output to the GFX Oscilloscopes, this can be considered a memory bottleneck because it is the memory access that is slowing down the performance of the application.

## **10 FUTURE OBJECTIVES**

### **10.1 OpenAS**

I would like to release this project as a free, open source library which people can use to implement vibration based touch surface technology into their applications with ease. I plan to call this library Open Audio Surface (OpenAS) and release it under the LGPL licence.

The advantage of this is that there will be no demand for the hardware unless developers start to integrate support for the technology into their applications. By doing this developers are encouraged to add support for this technology in their applications; essentially by supporting a wider range of hardware developers are opening their applications to a wider customer base.

### **10.2 Dedicated Hardware**

I would like to obtain a SHARC 32 bit floating point processor and evaluation board from Analog Devices, my intention is to have the SHARC processor dedicated to computing the entire math involved in detecting touch locations, interpolating intersample clips and so forth, so that all the end users computer receives are event messages notifying the computer when a touch has taken place, and what the details are such as force, position, etc. More accurate calculations could be computed on dedicated hardware such as the SHARC processor.

### **10.3 Plugin-Development**

I would like to release a series of plugins for software such as Max MSP, Processing, Cinder, VVV and other experimental prototyping languages not to forget hardware solutions too such as the Arduino and Beagle Board so that people can incorporate this technology into their home brew projects.

## **11 RECOMMENDATIONS**

### **11.1 Multi-Threading**

Processing the input of each microphone on a separate thread would tremendously improve the performance of this solution. Modern CPU's typically consist of four CPU cores, which is well suited for this type of application which processes the input of four microphones simultaneously.

### **11.2 Parallel Computing**

Parallel computing such as making use of available Single instruction, multiple data (SIMD) instructions in modules such as the touch detector would improve the performance of the module up to as much as four times. The SIMD instruction set allows four floating-point operations to be performed in one operation.

### **11.3 Time to Digital Converter**

Using a time to digital converter more accurate and configurable positions can be calculated, rather than using the amplitude as the delay input for the trilateration algorithm the time it takes for the touch vibration to get to each microphone could be used.

### **11.4 Laser Microphones**

Laser Microphones are a more practical method of measuring surface vibrations, currently using piezo electric contact microphones I require Blu-Tack to attach them to surfaces, while Blu-Tack is reusable and sticks to a wide range of surfaces, some very smooth surfaces Blu-Tack will not stick to, also the length of audio cable required grows as the size of the surface you want to use gets larger, using laser microphones you wouldn't have this problem. Cosmetically, it would look a lot cleaner too. The disadvantages of using laser microphones are that they would be incompatible with surfaces that exhibit transparency such as a glass window.

## 12 CONCLUSION

This project was a large risk and highly ambitious, having only hobbyist experience in digital signal processing and audio I decided to attempt something which to my knowledge had not been attempted prior, a lot of problems were encountered near the beginning where I had to re-design the entire system multiple times and as a result entered a large domain of research where I learnt a lot of new physics and mathematics.

The initial target for this project was to produce a technology which allows any surface to be turned into an accurate touch screen for use interacting with a computer, the resultant product is one that works but with a low degree of accuracy not suitable enough for commercial use as such it has been applied as a musical instrument where the loss of accuracy is acceptable; however this document outlines the techniques necessary to re-design the product into a more accurate solution, taking into consideration laser microphones, fixed stylus, time to digital converters and parallel processing.

This project was a prolonged research phase which produced a multiple of prototypes over the months, if the project had a longer time span the prototype would more advanced however I achieved and succeeded my initial outline all but in accuracy; having built an entire user interface system for debugging, real-time sound synthesis engine to apply it as a musical instrument and the development of signal interpolation techniques to improve the range and quality of data the application had work with.

The primary mistake in this project was to spend more time than necessary to develop an alternative audio buffering system; as a result it delayed the development of the solution by two or three weeks and in the end turned out to fragment the audio data rendering the buffering system useless; more time then had to be wasted on implementing a new buffering system.

## REFERENCES

Media College, "Condenser Microphones" [Online].

<http://www.mediacollege.com/audio/microphones/condenser.html>  
[07/04/2011]

Wikipedia, "Microphone" [Online].

<http://en.wikipedia.org/wiki/Microphone> [07/04/2011]

Realtek, "Realtek" [Online]. <http://www.realtek.com.tw/> [07/04/2011]

Creative, "Buy Creative Sound Blaster Sound Cards for PC" [Online].

<http://uk.store.creative.com/sound-blaster.aspx> [07/04/2011]

M-Audio, "M-AUDIO" [Online]. <http://www.m-audio.com/> [07/04/2011]

National Instruments, "National Instruments: Test, Measurement, and Embedded Systems" [Online]. <http://www.ni.com/> [07/04/2011]

Acquitek, "Data Acquisition Solutions - Acquitek" [Online].

<http://www.acquitek.com/data/acquisition-system-products.html>  
[07/04/2011]

Omega, "Data Acquisition Products" [Online].

<http://www.omega.com/pptst/das.html> [07/04/2011]

Gage, "Gage Applied" [Online]. <http://www.gage-applied.com/>

[07/04/2011]

SDL, "Simple DirectMedia Layer" [Online]. <http://www.libsdl.org/>

[07/04/2011]

PortAudio, "PortAudio - an Open-Source Cross-Platform Audio API"

[Online]. <http://www.portaudio.com/> [07/04/2011]

**Siddhant Ahuja, "Sum of Squared Differences" [Online].**

**<http://siddhantahuja.wordpress.com/tag/sum-of-squared-differences/>  
[07/04/2011]**

**The Engineering Toolbox, "Speed of Sound in some common Solids"  
[Online]. [http://www.engineeringtoolbox.com/sound-speed-solids-  
d\\_713.html](http://www.engineeringtoolbox.com/sound-speed-solids-d_713.html) [07/04/2011]**

**NDT, "The Speed of Sound in Other Materials" [Online]. [http://www.ndt-  
ed.org/EducationResources/HighSchool/Sound/speedinmaterials.htm](http://www.ndt-ed.org/EducationResources/HighSchool/Sound/speedinmaterials.htm)  
[07/04/2011]**

**DickBlick. (2006) [Online Image]**

**<http://cdn.dickblick.com/items/289/11/28911-1009-3ww-l.jpg> [Accessed  
08/04/2011]**

**ShopUK. (2009) [Online Image]**

**[http://www.shopuk.co.uk/lg\\_images/SMARTBoard\\_685\\_Interactive\\_Whit  
eboard.jpg](http://www.shopuk.co.uk/lg_images/SMARTBoard_685_Interactive_Whiteboard.jpg) [Accessed 08/04/2011]**

**GetPrice. (2011) [Online Image]**

**[http://www.getprice.com.au/images/uploadimg/319/350\\_\\_1\\_file\\_15\\_9.jpg](http://www.getprice.com.au/images/uploadimg/319/350__1_file_15_9.jpg)  
[Accessed 08/04/2011]**

**(2004) The Discrete Fourier Transform. In Lyons, R. Understanding  
Digital Signal Processing, pp. 45 - 145**

**(2004) The Fast Fourier Transform. In Lyons, R. Understanding Digital  
Signal Processing, pp. 125 - 151**

**(2004) Finite Impulse Response Filters. In Lyons, R. Understanding  
Digital Signal Processing, pp. 151 - 211**

**(2004) Infinite Impulse Response Filters. In Lyons, R. Understanding Digital Signal Processing, pp. 211 - 283**

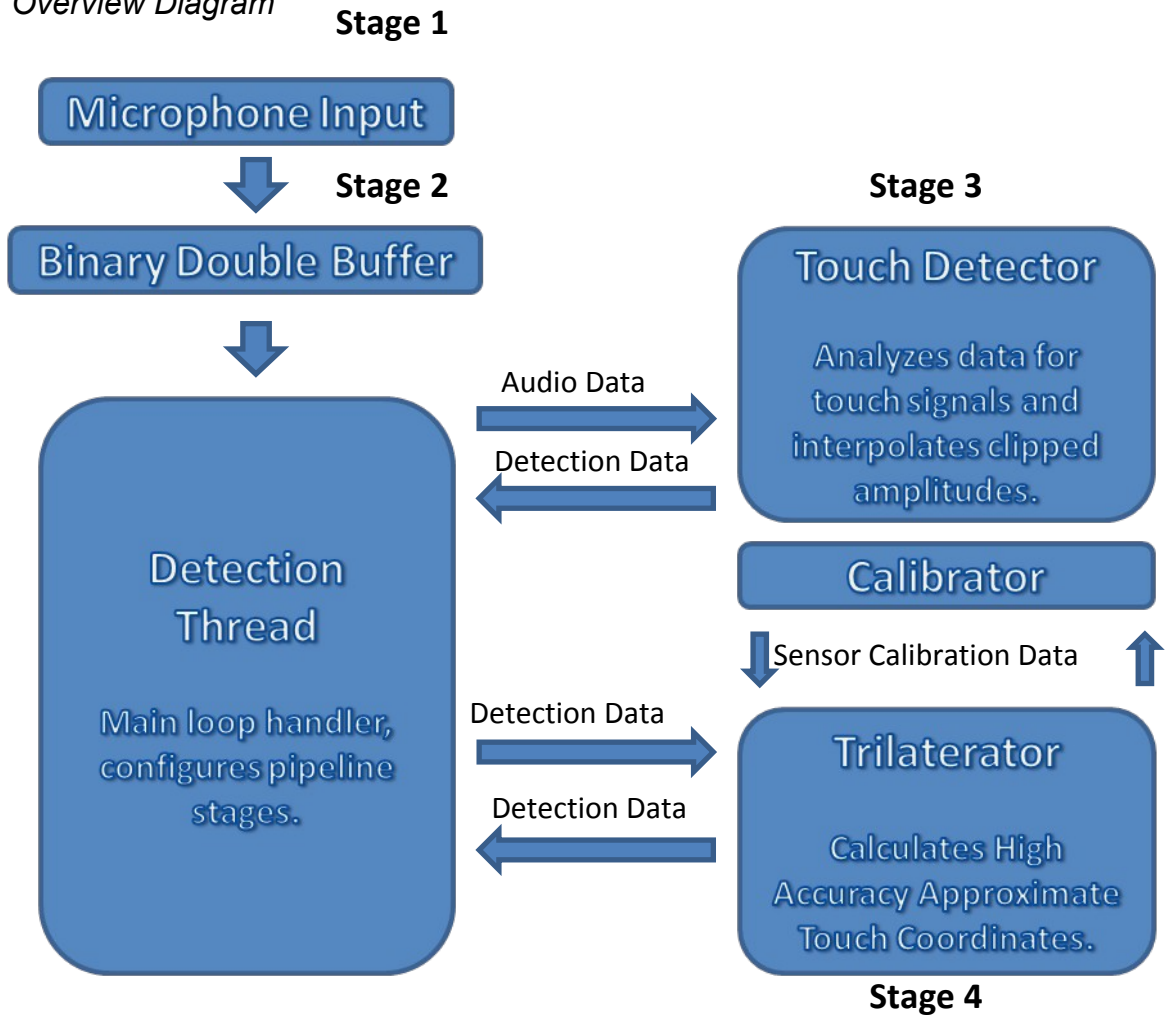
## **APPENDIX ASPECIFICATION**

The aim of this project is to investigate a method of providing touch screen projectors with the ability to measure the force at which you touched the projected image. At the moment most touch screen projector solutions use camera detection to locate where users have touched the projected image; this method is computationally expensive and also causes limitations such as being unable to measure the force that the projected image was touched with.

Contact microphones will be placed on the corners of the projected image; these microphones will detect the vibrations through the surface made by your contact with touching the surface. The delays between the microphones will be used in a trilateration algorithm to determine the position you touched the surface. The mean of waveform amplitudes from the contact microphones is used to calculate how hard you touched the surface.

## APPENDIX B PIPELINE DIAGRAMS

Overview Diagram



# APPENDIX C INTERSAMPLE INTERPOLATION FUNCTION

## REVISION 1

```
float DetectTouchForce(float* chunk, int size)
{
/*
    Code: James Fletcher 2011 - 2012
    Web: http://www.mrpuzzle.org/

    This function will return the highest amplitude in array specified.

    However it will also:
    1. Make a list of all intersample cut-offs
    2. Take the longest intersample cut-off from the list
    3. Detect the left and right slopes of the intersample cut-off
    4. Interpolate the intersample cut-off using available slopes and
    intersample cut-off length.
    5. Return interpolated value.

    Optimizations:
    Don't store all intersample cut-off's in a vector, use two buffers one
    to count intersample cut-offs and one to store the largest intersample
    cut-off found. If the buffer to count intersample cut-offs
    counts one that is longer than the one stored in the largest intersample
    buffer, update the largest intersample buffer with new intersample cut-
    off data.

    I assume that the wave is aligned to 0.f in a -1 to +1 scale.
*/

//If no amplitudes where cut-off, the highest amplitude is stored here and
returned.
float force = 0.f;

//Stores a list of detected intersample cut-offs
std::vector<int> m_cutoff_index;
std::vector<int> m_cutoff_len;
m_cutoff_index.reserve(5);
m_cutoff_len.reserve(5);

//Variables to check for cut-off amplitudes
int cutoff_startindex = 0;
int cutoff_len = 0;

//Loop through array and log amplitudes that where cut-off
for(int i = 0; i < size; ++i)
{
    //Check for the highest amplitude
    const float abschunk = fabs(chunk[i]);
    if(abschunk > force && abschunk <= 1.0f)
        force = abschunk;

    //If the amplitude is within the cut-off get ready to count a possible
    intersample cut-off
    if(chunk[i] >= 0.94f && chunk[i] <= 1.0f) //I don't need to consider the
    negative scale (other wise we may over count intersample cut-offs)
    {
        if(cutoff_startindex == 0)
            cutoff_startindex = i;
```

```

        if(cutoff_startindex != 0)
            cutoff_len++;
    }
    else
    {
        //Was this an intersample cut-off?
        if(cutoff_len > 1)
        {
            //YES it was! Store Intersample Cut-off Info
            m_cutoff_index.push_back(cutoff_startindex);
            m_cutoff_len.push_back(cutoff_len);

            //Get ready to detect another intersample cut-off
            cutoff_startindex = 0;
            cutoff_len = 0;
        }
        else
        {
            //No it wasn't, it was just a single sample
            if(cutoff_len != 0)
                force = 1.0f;
        }
    }
}

//If there were no recorded intersample cut-off's, return the max amplitude
//recorded
const int arraysize = m_cutoff_index.size();
if(arraysize == 0)
    return force;

//Otherwise loop through array and get the largest intersample cut-off
int index = 0;
for(int i = 0; i < arraysize; ++i)
{
    if(m_cutoff_len[index] < m_cutoff_len[i])
        index = i;
}

//Storage for index points of left slope and right slope of intersample cut-off
int left_index = -1, right_index = -1;

//Storage for left slope and right slope rates of change
float left_reduction = 0.f, right_reduction = 0.f;

//Find the start index of the left slope
float last_amp = 1.0f;
for(int i = m_cutoff_index[index]; chunk[i] <= last_amp && i >= 0; --i) //Stop
when last value is smaller than the current one
{
    left_index = i;
    last_amp = chunk[i];
}

//Find the end index of the right slope
last_amp = 1.0f;
for(int i = m_cutoff_index[index] + m_cutoff_len[index]; chunk[i] <= last_amp
&& i < size; ++i) //Stop when last value is smaller than the current one
{
    right_index = i;
    last_amp = chunk[i];
}

```

```

}

//Workout Average Reduction in Amplitude for left slope (rate of change)
if(left_index != -1) //Make sure we actually found the index of slope
{
    for(int i = left_index+1; i < m_cutoff_index[index]; i += 2)
    {
        if(i-1 >= 0) //Don't over run the buffer!!
            left_reduction += chunk[i-1] - chunk[i];
    }
    if(left_reduction != 0.f)
        left_reduction /= m_cutoff_index[index] - (left_index+1);
}

//Workout Average Reduction in Amplitude for right slope (rate of change)
if(right_index != -1) //Make sure we actually found the index of slope
{
    for(int i = right_index-1; i > m_cutoff_index[index] +
        m_cutoff_len[index]; i -= 2)
    {
        if(i+1 < size) //Don't over run the buffer!!
            right_reduction += chunk[i] - chunk[i+1];
    }
    if(right_reduction != 0.f)
        right_reduction /= (right_index+1) - (m_cutoff_index[index] +
            m_cutoff_len[index]);
}

//If the left slope was not available in sample
if(left_reduction == 0.f && right_reduction != 0.f)
{
    //Precompute half the size of the cut-off length
    const float halflen = (m_cutoff_len[index] * 0.5f);

    //Return max amp of the right slope
    return fabs(1.0f + ((chunk[m_cutoff_index[index]+m_cutoff_len[index]+1]
        * halflen) - (right_reduction * halflen)));
}

//If the right slope was not available in sample
if(right_reduction == 0.f && left_reduction != 0.f)
{
    //Precompute half the size of the cut-off length
    const float halflen = (m_cutoff_len[index] * 0.5f);

    //Return max amp of the left slope
    return fabs(1.0f + ((chunk[m_cutoff_index[index]-1] * halflen) -
        (left_reduction * halflen)));
}

//If both slopes where available
if(left_reduction != 0.f && right_reduction != 0.f)
{
    //Precompute half the size of the cutoff length
    const float halflen = (m_cutoff_len[index] * 0.5f);

    //Get Approximate Max Amplitude from left slope
    const float maxamp_left = 1.0f + ((chunk[m_cutoff_index[index]-1] *
        halflen) - (left_reduction * halflen));

    //Get Approximate Max Amplitude from right slope
    const float maxamp_right = 1.0f +

```

```

        ((chunk[m_cutoff_index[index]+m_cutoff_len[index]+1] * halflen) -
         (right_reduction * halflen));

        //Return the average of the two!
        return fabs((maxamp_left + maxamp_right) * 0.5f);
    }

    //Bugger!! No slopes where available!!
    if(left_reduction == 0.f && right_reduction == 0.f)
    {
        //Times the cutoff len by two, best approximation that can be done with
        no slopes!!
        return m_cutoff_len[index] * 2.f;
    }

    //Urge this will never happen... But to satisfy the compiler...
    return 1.0f;
}

```

## REVISION 2

```

//Detect the contact force
float DetectTouchForce(float* chunk, int size)
{
    /*
        Code: James Fletcher 2011 - 2012
        Web: http://www.mrpuzzle.org/

        This function will return the highest amplitude in array specified.

        However it will also:
        1. Make a list of all intersample cut-offs
        2. Take the longest intersample cut-off from the list
        3. Detect the left and right slopes of the intersample cut-off
        4. Interpolate the intersample cut-off using available slopes and
        intersample cut-off length.
        5. Return interpolated value.

        Optimizations:
        Don't store all intersample cut-off's in a vector, use two buffers one
        to count intersample cut-offs and one to store the largest intersample
        cut-off found. If the buffer to count intersample cut-offs
        counts one that is longer than the one stored in the largest intersample
        buffer, update the largest intersample buffer with new intersample cut-
        off data.

        I assume that the wave is aligned to 0.f in a -1 to +1 scale.
    */

    //If no amplitudes where cutoff, the highest amplitude is stored here and
    returned.
    float force = 0.f;

    //Stores a list of detected intersample cutoffs
    std::vector<int> m_cutoff_index;
    std::vector<int> m_cutoff_len;
    m_cutoff_index.reserve(5);
    m_cutoff_len.reserve(5);

    //Variables to check for cutoff amplitudes

```

```

int cutoff_startindex = 0;
int cutoff_len = 0;

//Loop through array and log amplitudes that where cutoff
for(int i = 0; i < size; ++i)
{
    //Check for the highest amplitude
    const float abschunk = fabs(chunk[i]);
    if(abschunk > force && abschunk <= 1.0f)
        force = abschunk;

    //If the amplitude is within the cutoff get ready to count a possible
    intersample cutoff
    if(chunk[i] >= 0.90f && chunk[i] <= 1.0f)
    {
        if(cutoff_startindex == 0)
            cutoff_startindex = i;

        if(cutoff_startindex != 0)
            cutoff_len++;
    }
    else
    {
        //Was this an intersample cutoff?
        if(cutoff_len > 1)
        {
            //YES it was! Store Intersample Cutoff Info
            m_cutoff_index.push_back(cutoff_startindex);
            m_cutoff_len.push_back(cutoff_len);

            //Get ready to detect another intersample cutoff
            cutoff_startindex = 0;
            cutoff_len = 0;
        }
        else
        {
            //No it wasnt, it was just a single sample
            if(cutoff_len != 0)
                force = 1.0f;
        }
    }
}

//If there where no recorded intersample cutoff's, return the max amplitude
recorded
const int arraysize = m_cutoff_index.size();
if(arraysize == 0)
    return force;

//Otherwise loop through array and get the largest intersample cutoff
int index = 0;
for(int i = 0; i < arraysize; ++i)
{
    if(m_cutoff_len[index] < m_cutoff_len[i])
        index = i;
}

//Storage for index points of ends of left slope and right slope of intersample
cutoff
int left_index = -1, right_index = -1;

//Storage for left slope and right slope rates of change

```

```

float left_reduction = 0.f, right_reduction = 0.f;

//Sotrage for avergae left and right slope increments
float left_increment = 0.f, right_increment = 0.f;

//Find the start index of the left slope
float last_amp = 1.0f;
for(int i = m_cutoff_index[index]; chunk[i] <= last_amp && i >= 0; --i) //Stop
when last value is smaller than the current one
{
    left_index = i;
    last_amp = chunk[i];
}

//Find the end index of the right slope
last_amp = 1.0f;
for(int i = m_cutoff_index[index] + m_cutoff_len[index]; chunk[i] <= last_amp
&& i < size; ++i) //Stop when last value is smaller than the current one
{
    right_index = i;
    last_amp = chunk[i];
}

//Workout Average Reduction in Amplitude for left slope (rate of change)
int count = 0;
if(left_index != -1) //Make sure we actually found the index of slope
{
    for(int i = left_index+1; i < m_cutoff_index[index]; i += 2)
    {
        if(i-1 >= 0) //Don't over run the buffer!!
        {
            count++;
            left_increment += fabs(chunk[i-1]) + fabs(chunk[i]);
            left_reduction += chunk[i-1] - chunk[i];
        }
    }
    if(left_reduction != 0.f)
    {
        if(count > 1)
        {
            const float recip = 1.f / count;
            left_increment *= recip;
            left_reduction *= recip;
        }
    }
}

//Workout Average Reduction in Amplitude for right slope (rate of change)
count = 0;
if(right_index != -1) //Make sure we actually found the index of slope
{
    for(int i = right_index-1; i > m_cutoff_index[index] +
m_cutoff_len[index]; i -= 2)
    {
        if(i+1 < size) //Don't over run the buffer!!
        {
            count++;
            right_increment += fabs(chunk[i]) + fabs(chunk[i+1]);
            right_reduction += chunk[i] - chunk[i+1];
        }
    }
    if(right_reduction != 0.f)

```

```

    {
        if(count > 1)
        {
            const float recip = 1.f / count;
            right_increment *= recip;
            right_reduction *= recip;
        }
    }
}

//If the left slope was not available in sample
if(left_reduction == 0.f && right_reduction != 0.f)
{
    //Precompute half the size of the cutoff length
    const float halflen = (m_cutoff_len[index] * 0.5f);

    //Return max amp of the right slope
    return fabs(1.0f + ((right_increment * halflen) - (right_reduction *
    halflen)));
}

//If the right slope was not available in sample
if(right_reduction == 0.f && left_reduction != 0.f)
{
    //Precompute half the size of the cutoff length
    const float halflen = (m_cutoff_len[index] * 0.5f);

    //Return max amp of the left slope
    return fabs(1.0f + ((left_increment * halflen) - (left_reduction *
    halflen)));
}

//If both slopes were available
if(left_reduction != 0.f && right_reduction != 0.f)
{
    //Precompute half the size of the cutoff length
    const float halflen = (m_cutoff_len[index] * 0.5f);

    //Get Approximate Max Amplitude from left slope
    const float maxamp_left = 1.0f + ((left_increment * halflen) -
    (left_reduction * halflen));

    //Get Approximate Max Amplitude from right slope
    const float maxamp_right = 1.0f + ((right_increment * halflen) -
    (right_reduction * halflen));

    //Return the average of the two!
    return fabs((maxamp_left + maxamp_right) * 0.5f);
}

//Bugger!! No slopes were available!!
if(left_reduction == 0.f && right_reduction == 0.f)
{
    //Times the cutoff len by two, best approximation that can be done with
    no slopes!!
    return m_cutoff_len[index];
}

//Urgh this will never happen.. But incase it does..
return 1.0f;
}

```

## APPENDIX DBINARY DOUBLE BUFFER CLASS

### HEADER

```
#ifndef BINARYDOUBLEBUFFER_H
#define BINARYDOUBLEBUFFER_H

#include "Globals.h"
#include <stdio.h>

//This will effect responciveness
#define CHUNK_SIZE 128
static const int CHUNK_SIZE_FLOAT = CHUNK_SIZE * sizeof(float);
static const int CHUNK_SIZE_X2 = CHUNK_SIZE * 2;
static const int CHUNK_SIZE_X3 = CHUNK_SIZE * 3;

//MultiBuffer Chunk
struct Chunk
{
    Chunk() : isold(true) {} //constructor
    float chunk[CHUNK_SIZE]; //chunk data
    bool isold; //has it been processed
};

//BinaryDoubleBuffer Class (maximum possible delay is < 90ms)
class BinaryDoubleBuffer
{
public:

    //Add a chunk to the multi buffer
    bool AddChunk(const float* data);

    //Get a chunk from the multi buffer
    bool GetChunk(float* data);

    //Sample Counter
    void StartSampleCounter(int startoffset);
    int GetElapsedSamples();
    void EndSampleCounter();

private:

    //Do Functions
    void DoAddChunk(int i, const float* data);
    void DoGetChunk(int i, const float* data);

    //Chunk Buffer
    Chunk m_chunks[30];

    //sizeof(float) * CHUNK_SIZE
    static int floatxchunksiz;

    //Sample Counting
    int m_countedsamples;
    bool m_counting;

};

MP_INLINE void BinaryDoubleBuffer::DoAddChunk(int i, const float* data)
```

```

{
    m_chunks[i].isold = false;
    memcpy(m_chunks[i].chunk, data, floatxchunksize);

#ifdef _DEBUG
    m_recorded++;
#endif
}

MP_INLINE void BinaryDoubleBuffer::DoGetChunk(int i, const float* data)
{
    m_chunks[i].isold = true;
    memcpy((float*)data, m_chunks[i].chunk, floatxchunksize);
}

MP_INLINE void BinaryDoubleBuffer::StartSampleCounter(int startoffset)
{
    m_countedsamples = startoffset;
    m_counting = true;
}

MP_INLINE void BinaryDoubleBuffer::EndSampleCounter()
{
    m_counting = false;
}

MP_INLINE int BinaryDoubleBuffer::GetElapsedSamples()
{
    return m_countedsamples; //An end offset could not be accurately
                             found as streams are not parallelised.
}

#endif

```

## SOURCE

```

#include "BinaryDoubleBuffer.h"

//Set static member variable
int BinaryDoubleBuffer::floatxchunksize = sizeof(float) * CHUNK_SIZE;

//Add a chunk to the multi buffer
bool BinaryDoubleBuffer::AddChunk(const float* data)
{
    if(m_counting == true)
        m_countedsamples += CHUNK_SIZE;

    if(m_chunks[0].isold == true)
    {
        if(m_chunks[2].isold == true)
        {
            if(m_chunks[6].isold == true)
            {
                if(m_chunks[14].isold == true)
                {
                    DoAddChunk(14, data);
                    return true;
                }
            }
        }
    }
}

```

```

        if(m_chunks[15].isold == true)
        {
            DoAddChunk(15, data);
            return true;
        }

        DoAddChunk(6, data);
        return true;
    }

    if(m_chunks[7].isold == true)
    {

        if(m_chunks[16].isold == true)
        {
            DoAddChunk(16, data);
            return true;
        }

        if(m_chunks[17].isold == true)
        {
            DoAddChunk(17, data);
            return true;
        }

        DoAddChunk(7, data);
        return true;
    }

    DoAddChunk(2, data);
    return true;
}

if(m_chunks[3].isold == true)
{

    if(m_chunks[8].isold == true)
    {

        if(m_chunks[18].isold == true)
        {
            DoAddChunk(18, data);
            return true;
        }

        if(m_chunks[19].isold == true)
        {
            DoAddChunk(19, data);
            return true;
        }

        DoAddChunk(8, data);
        return true;
    }

    if(m_chunks[9].isold == true)
    {

        if(m_chunks[20].isold == true)
        {
            DoAddChunk(20, data);

```

```

        return true;
    }

    if(m_chunks[21].isold == true)
    {
        DoAddChunk(21, data);
        return true;
    }
    DoAddChunk(9, data);
    return true;
}

DoAddChunk(3, data);
return true;
}

DoAddChunk(0, data);
return true;
}

if(m_chunks[1].isold == true)
{
    if(m_chunks[4].isold == true)
    {
        if(m_chunks[10].isold == true)
        {
            if(m_chunks[22].isold == true)
            {
                DoAddChunk(22, data);
                return true;
            }

            if(m_chunks[23].isold == true)
            {
                DoAddChunk(23, data);
                return true;
            }

            DoAddChunk(10, data);
            return true;
        }

        if(m_chunks[11].isold == true)
        {
            if(m_chunks[24].isold == true)
            {
                DoAddChunk(24, data);
                return true;
            }

            if(m_chunks[25].isold == true)
            {
                DoAddChunk(25, data);
                return true;
            }

            DoAddChunk(11, data);
            return true;
        }
    }
}

```

```

        DoAddChunk(4, data);
        return true;
    }

    if(m_chunks[5].isold == true)
    {
        if(m_chunks[12].isold == true)
        {
            if(m_chunks[26].isold == true)
            {
                DoAddChunk(26, data);
                return true;
            }

            if(m_chunks[27].isold == true)
            {
                DoAddChunk(27, data);
                return true;
            }

            DoAddChunk(12, data);
            return true;
        }

        if(m_chunks[13].isold == true)
        {
            if(m_chunks[28].isold == true)
            {
                DoAddChunk(28, data);
                return true;
            }

            if(m_chunks[29].isold == true)
            {
                DoAddChunk(29, data);
                return true;
            }

            DoAddChunk(13, data);
            return true;
        }

        DoAddChunk(5, data);
        return true;
    }

    DoAddChunk(1, data);
    return true;
}

return false;
}

//Get a chunk from the multi buffer
bool BinaryDoubleBuffer::GetChunk(float* data)
{
    if(m_chunks[0].isold == false)

```

```

{
    if(m_chunks[2].isold == false)
    {
        if(m_chunks[6].isold == false)
        {
            if(m_chunks[14].isold == false)
            {
                DoGetChunk(14, data);
                return true;
            }

            if(m_chunks[15].isold == false)
            {
                DoGetChunk(15, data);
                return true;
            }

            DoGetChunk(6, data);
            return true;
        }

        if(m_chunks[7].isold == false)
        {
            if(m_chunks[16].isold == false)
            {
                DoGetChunk(16, data);
                return true;
            }

            if(m_chunks[17].isold == false)
            {
                DoGetChunk(17, data);
                return true;
            }

            DoGetChunk(7, data);
            return true;
        }

        DoGetChunk(2, data);
        return true;
    }

    if(m_chunks[3].isold == false)
    {
        if(m_chunks[8].isold == false)
        {
            if(m_chunks[18].isold == false)
            {
                DoGetChunk(18, data);
                return true;
            }

            if(m_chunks[19].isold == false)
            {
                DoGetChunk(19, data);
            }
        }
    }
}

```

```

        return true;
    }

    DoGetChunk(8, data);
    return true;
}

if(m_chunks[9].isold == false)
{
    if(m_chunks[20].isold == false)
    {
        DoGetChunk(20, data);
        return true;
    }

    if(m_chunks[21].isold == false)
    {
        DoGetChunk(21, data);
        return true;
    }

    DoGetChunk(9, data);
    return true;
}

DoGetChunk(3, data);
return true;
}

DoGetChunk(0, data);
return true;
}

if(m_chunks[1].isold == false)
{
    if(m_chunks[4].isold == false)
    {
        if(m_chunks[10].isold == false)
        {
            if(m_chunks[22].isold == false)
            {
                DoGetChunk(22, data);
                return true;
            }

            if(m_chunks[23].isold == false)
            {
                DoGetChunk(23, data);
                return true;
            }

            DoGetChunk(10, data);
            return true;
        }

        if(m_chunks[11].isold == false)
        {

```

```

        if(m_chunks[24].isold == false)
        {
            DoGetChunk(24, data);
            return true;
        }

        if(m_chunks[25].isold == false)
        {
            DoGetChunk(25, data);
            return true;
        }

        DoGetChunk(11, data);
        return true;
    }

    DoGetChunk(4, data);
    return true;
}

if(m_chunks[5].isold == false)
{
    if(m_chunks[12].isold == false)
    {
        if(m_chunks[26].isold == false)
        {
            DoGetChunk(26, data);
            return true;
        }

        if(m_chunks[27].isold == false)
        {
            DoGetChunk(27, data);
            return true;
        }

        DoGetChunk(12, data);
        return true;
    }

    if(m_chunks[13].isold == false)
    {
        if(m_chunks[28].isold == false)
        {
            DoGetChunk(28, data);
            return true;
        }

        if(m_chunks[29].isold == false)
        {
            DoGetChunk(29, data);
            return true;
        }

        DoGetChunk(13, data);
        return true;
    }

    DoGetChunk(5, data);
}

```

```
        return true;
    }
    DoGetChunk(1, data);
    return true;
}
return false;
}
```

## APPENDIX F PROJECT DIARIES

Name: James Fletcher  
Supervisor: Jay Chapman  
Date: 20/10/2010

### Issues identified in previous meeting (including personal development goals):

Project Methodology needs to be chosen.

Report template and outlines need to be started.

Project Specification needs to be submitted.

### Feedback received in previous meeting:

This is the first meeting.

### Action taken on feedback:

I have chosen to use the iterative / incremental methodology for this project; I looked at a range of methodologies that I narrowed down to Spiral, Agile, Waterfall, and Iterative. I chose iterative because agile was too team/people oriented for this project, waterfall was too limiting by not allowing re-iterating on previous chunks of the project and the spiral model seemed more suited to larger projects with more risk. The iterative / incremental model allows me to complete the project in linear chunks while allowing me to go back and re-iterate on chunks if need be.

Project specification has been finalised and submitted.

### Matters to discuss:

Using Microsoft word in outline mode

Next stage of project

### References Consulted:

<http://www.ibm.com/developerworks/rational/library/may05/bittner-spence>

<http://www.ibm.com/developerworks/rational/library/4626.html>

[http://en.wikipedia.org/wiki/Spiral\\_model](http://en.wikipedia.org/wiki/Spiral_model)

[http://en.wikipedia.org/wiki/Iterative\\_and\\_incremental\\_development](http://en.wikipedia.org/wiki/Iterative_and_incremental_development)

<http://stackoverflow.com/questions/3897335/whats-the-difference-between-incremental-software-process-model-evolutionary-mo>

[http://en.wikipedia.org/wiki/Software\\_development\\_methodologies#Waterfall\\_Approach](http://en.wikipedia.org/wiki/Software_development_methodologies#Waterfall_Approach)

<http://www.mariosalexandrou.com/methodologies.asp>

Name:	James Fletcher
Supervisor:	Jay Chapman
Date:	29/10/2010

**Issues identified in previous meeting (including personal development goals):**

Project Prototype needs to be developed and tested before any further research.

**Action taken on feedback:**

I have bought all of the hardware components and selected the C++ libraries that I will be using the interface with the hardware after some research. I will be creating a prototype on a fixed surface for testing purposes that will take input from the contact microphones and use a form of trilateration to calculate the vibration origin. Once this has been achieved current research can be taken from notes to full detail and further research can begin.

Name: James Fletcher  
Supervisor: Jay Chapman  
Date: 3/12/2010

**Issues identified in previous meeting (including personal development goals):**

The solution I have developed has a sampling rate and timing accuracy that is too slow to accurately detect the delays between sounds on a small scale. I am sampling sound through a microphone at 44100 Hz which means I only get 196 sample points at microsecond accuracy. That's 196 sample points out of 1000. Sound travels roughly 1mm every 156 nanoseconds so I would need to sample at nanosecond accuracy.

**Action taken on feedback:**

I have contacted Analog Devices about their analog to digital converters, I will see what solutions they have available, it would be more efficient it also get a 32 bit floating point processor from them to do the calculations on so that all I send to the computer is notifications when a touch has been registered etc., a computer is capable of nanosecond timing but a lot of accuracy and efficiency would be lost due to the event driven nature of operating systems.

The solution is now a bit more complicated than some cheap USB sound cards and a piece of OS level software.

**Matters to discuss:**

Where to take my current inaccurate working solution

How to fund the analog to digital converter and floating point processor

Name: James Fletcher  
Supervisor: Jay Chapman  
Date: 27/12/2010

**Issues identified in previous meeting (including personal development goals):**

Sampling Frequency of soundcard analog inputs was too low to detect sound delays within a satisfactory range through solid objects.

**Action taken on feedback:**

I have intensely researched data acquisition, such as analog to digital converters and digitizers. I have received quotes, one particular from TTI who quoted me a range between £3000 and £5000 before tax for digitizer with the capability to sample at a rate of around 100mhz per channel across four channels. I also had a quote of £1200 for a digitizer that has a sampling frequency of 20mhz.

**Matters to discuss:**

Obviously I cannot fund this, but possibly the university could invest in these facilities as data acquisition can be applied to a wide range of computing applications. They may very well already provide these facilities and I will have to investigate this.

I have decided for now I will simply limit the current working prototype to a specific surface that is a base of solid plastic, then a layer of polystyrene and then a layer on cling film in an attempt to slow the speed of the vibrations so that the 44khz sampling rate is sufficient in distances of 1mm to 1cm.

Also I will be looking to expand the scope of the project into the area of touch screen technology using webcams to simulate brush strokes. This 'fork' if you will has less technical background research as it is a relatively simple idea but will have plentiful research into materials and textures.

**References Consulted:**

[http://www.mccdaq.com/pci-data-acquisition/PCI-DAS\\_4020-12.aspx](http://www.mccdaq.com/pci-data-acquisition/PCI-DAS_4020-12.aspx)

Quote from TTI (uk):

Aquitek PM66-14 HSA 14 £3470.39

Gage Octopus CS8240 Compuscope £2894.74

Gage Octopus CS8242 Compuscope £3348.68

Gage Octopus CS8244 Compuscope £3802.63

Gage Octopus CS8245 Compuscope £4256.58

Gage Octopus CS8247 Compuscope £4710.53

Gage Octopus CS8249 Compuscope £5164.47

Name: James Fletcher  
Supervisor: Jay Chapman  
Date: 27/12/2010

**Issues identified in previous meeting (including personal development goals):**

Decided to reduce the speed the sound travels through test surface rather than sample at a higher frequency to keep the costs down.

**Action taken on feedback:**

I have done some research into what determines the speed that sound travels through a material. I have found that elasticity is the biggest factor that determines the speed a sound travels, the more elastic the bond between particles (the more solid the object) the faster sound waves can travel between particles. The second factor is the density of particles as it takes more energy to make bigger (more dense) particles vibrate. Rubbers reduce the speed of sound down to around 60 m/s which is six times slower than how fast sound travels through air. I researched into brush-on Utherane and Silicone Rubber but in the end have decided to just to buy rubber sheeting which I will stick to the surface of my plastic chopping board.

By using rubber as a surface it allows me to engrave the surface with a range of different textures.

**Matters to discuss:**

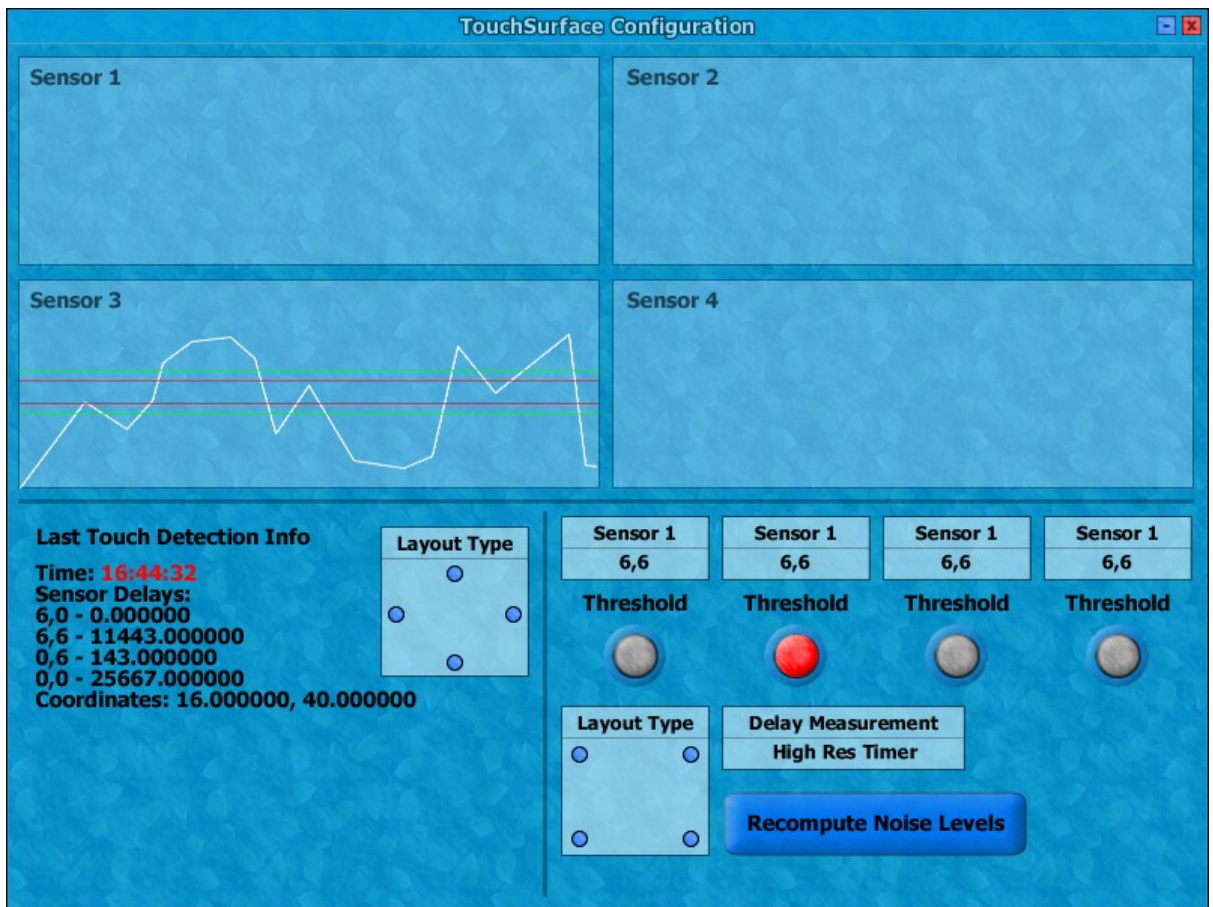
Should I do more research into the area of rubbers, and their particle level bonds?

Name: James Fletcher  
Supervisor: Jay Chapman  
Date: 27/12/2010

**Issues identified in previous meeting (including personal development goals):**

Researched rubber

**Action taken on feedback:**



I have started work on the user interface.

## **APPENDIX G      QUESTIONNAIRES**

Questionnaires follow after this page.