**Developing an Authentic, Flexible**

**Multi-touch Percussion Instrument**

Thesis Proposal

By John Cook

**Contents**

Abstract 4

I Introduction 5

Importance of Electronic Instruments 5

Problems with Traditional Electronic Percussion Instruments 6

Scope 7

Proposed Solution 7

Evaluation of Results 8

Required Resources 8

Description of Remaining Sections 9

II Previous Work and Application 9

Multi-Touch Hardware 10

Projection 13

Multi-Touch APIs and Frameworks 14

III Methods Used

IV Conclusion

Glossary

Appendix A: User Manual

Appendix B: Maintenance Manual

Appendix C: Source Code

Appendix D: Requirements Document

**Table of Figures**

Figure I TUIO Client/Tracker Application Diagram 16

**Abstract**

Electronic musical instruments have many benefits over their acoustic counterparts, but they have always been lacking in certain areas. To performers, the most notable of these areas is an authentic performance experience.

This report presents a new method of design for an electronic hand percussion instrument, using multi-touch technology, to create authentic interaction with the instrument. The name of the program is HanDrum. The performer can interact with the instrument in the same manner as that of its acoustic counterpart, while still reaping the benefits of the electronic instrument.

A prototype will been developed in this project to test whether this method of authentic interaction is possible. This report states whether this is possible based on the results of the program development.

**I Introduction**

This report suggests the design of an electronic percussion instrument with interaction capabilities similar to that of its acoustic counterpart. A software-based instrument will be developed in order to determine if, and to what extent, it is possible to create an electronic instrument with such capabilities using multi-touch technology in its physical interface.

**Importance of Electronic Instruments**

Electronic music has come a long way since Robert Moog built the first electronic synthesizer in 1964. Software- and hardware-based electronic instruments have become a stable, even a necessity, in today’s music industry. Originally, these instruments were designed in order to produce sounds that could not be created or re-created on acoustic instruments. While the range of tones that could be produced varied wildly, from smooth and soothing to harsh and jarring, they were all undeniably electronic. Similarly, when Les Paul mounted electric pickups on a solid guitar body he changed the sound of music forever.

As electronic instruments began to develop, their implementation began to spread. While synthesizers and guitars were the first electronic instruments, today we have an electric variation of almost every acoustic instrument imaginable. Ultimately, today’s electronic instruments provide two outstanding benefits over their acoustic counterparts. First, they provide the ability to produce sounds that the acoustic instrument is not able to produce (guitar, saxophone, etc.). Second, in some cases, they provide a degree of convenience or portability not afforded by the acoustic instrument (drums, upright bass, etc.).

**Problems with Traditional Electronic Percussion Instruments**

As the hardware and software makeup of electronic instruments has changed, the desire of musicians to create ever more realistic electronic versions of commonly acoustic instruments has not. With software-based physical modeling and high quality recorded samples, modern electronic instruments have the ability to produce lifelike sounds that are authentic to even the most trained ear. The way that we physically interact with these instruments, however, has not seen this “Moore’s Law” type of innovation.

Currently, all electronic percussion instruments use the same technology for physical input. That technology consists of a sensor (typically a piezo) connected to a striking surface which, when struck, sends a signal to a piece of computer software. This signal causes the software to play a sample that corresponds with the type of signal that was received. The software determines the velocity of the strike based on the input signal, and the placement of the strike based on which sensor the signal came from. Some companies have tried to improve the performer’s interaction experience by changing the striking surface to a mesh that closely resembles an acoustic drum head.

Despite the authentic feel of these new mesh drum heads, all acoustic percussion instruments share the same set of problems: the inability to determine different types of strikes/strokes, and the lack of an appropriate sample set to correspond to these strikes/strokes. While velocity and placement give a good indication of how the performed intended the instrument to sound, these measures are simple not enough to determine the intended result.

For instance, there are two methods of playing a rim shot. One involves striking the drum head and rim at the same time. The other is played by placing the tip of a drum stick on the head and resting the shaft of the stick on the rim, then striking the stick with another drum stick. While these methods produce discernibly different sounds, they would be interpreted by a conventional electronic drum as being the same.

Another issue is the cross stick. This is played by resting the tip of a drum stick on the drum head and “clicking” it against the rim. In this case, the sensor may or may not pick up the small vibration against the head that helps to create this signature sound. Some electronic drums are designed to be sensitive enough to pick up this technique, albeit unreliably, while others lack this ability altogether.

**Scope**

It should be possible to solve the problem above if it is possible to create a method of input that can determine the intention of the performer by determining different types of strikes/strokes, and by including an appropriate sample set to correspond to these strikes/strokes, while maintaining an authentic performance experience. In addition, this instrument will potentially have the ability to recognize new forms of interaction. This ability will increase its relevance in the electronic percussion industry, as well as the electronic instrument industry by and large.

**Proposed Solution**

This report suggests the design of a software-based hand percussion instrument that utilizes optical multi-touch technology for its physical interface. The program will use the commonly accepted “zone” system to determine velocity, and will use fiducial placement and number to interpret the intent of the performer.

**Evaluation of Results**

The results of this project must be evaluated against two metrics. First, the instrument will be evaluated against other electronic percussion instruments in order to determine what, if any, improvement over existing design and technology has been made. Second, the instrument will be evaluated for authenticity of interaction against its acoustic counterpart. This second test will hold more weight than the first as the it is the object of this project to reproduce the interaction of the acoustic instrument.

The instrument will be evaluated by percussion performers who have experience with industry standard percussion instruments as well as acoustic hand drums. While it is not possible to correlate any hard data to these comparisons, the performers will use standard acoustic drum technique when evaluating the instrument. As such, they will be able to determine if, and in what capacities, the instrument is identically playable to its acoustic counterpart.

As for comparison to competitors, a system of certain criteria to be met may be created. This could include response to certain velocity ranges, interpreting intentions between similar stoke methods, and responding accurately to differing striking methods (hands, fingers, beaters, etc.). These criteria could be charter as metrics, allowing the instrument and its competitors to be evaluated in a quantifiable manner.

**Required Resources**

The resources required for this project can be placed into two categories: software and hardware. A computer running an integrated development environment (IDE) will be necessary in order to write the code for the software. A knowledge of the Python programming language will also be necessary. The materials for the hardware build are as follows: wood, carpentry tools (hammer, nails, etc.), one sheet of 1/4” thick plexi-glass, one sheet of silicon, and one infrared LED array. Various materials may be needed to mount the LED array around the plexi-glass, but these materials are unknown at this time. A graphical user interface may be included based on software development and materials cost. This may provided by a liquid crystal display (LCD) panel or projector mounted underneath the plexi-glass. A work space will be needed to complete this build.

**Overview of Remaining Sections**

Section Two will outline all previous work related to, and how it will (or will not) be applied to this project. Section Three will outline the methods used in the design of this project. Section Four will conclude this report by presenting the results of, and by discussing the problems left unsolved by, the project.

**II Previous Work and Application**

There are fine examples of electronic instruments whose physical interaction mimics that of their acoustic counterpart. Electronic pianos are the most obvious example of this type of instrument. Brian Moore’s iGuitar, Yamaha’s Wx5 Wind Controller, and Pearl’s e-Pro Live drum set are examples of electronic instruments that are typically thought of as having acoustic playability. While a few manufacturers are producing electronic hand drums, this market segment has thus far been left behind in the progression of realistic interaction. While there are a few examples of electronic hand drums on the market such as Roland’s HandSonic, Korg’s WaveDrum percussion synthesizer is the most realistically played model available. This unit, however, is limited by its built in sound bank and ability to only be physically played by traditional methods. Also, despite having best-in-class playability, it is still lacking when compared to an acoustic hand drum. As previously stated, it is within the scope of this project to expound on the authentic nature of interaction with the suggested program. The WaveDrum falls short of this ability as well in its limitation to traditional means of interaction.

**Multi-Touch Hardware**

There are two types of multi-touch technology in use today: capacitive and optical. Capacitive technology uses near-microscopic capacitors to register the contact and location of a capacitive conductor with the screen. Due to the high development cost of capacitive touch surfaces, optical multi-touch has been selected as the technology that this instrument will be based off of.

There are six forms of optical multi-touch technology in use today, each with its own unique advantages and disadvantages. These technologies are all similar in that they consist of a glass touch surface and a camera to detect when a touch is made. In any of these cases, when an object touches the surface is reflects more light toward the camera than the area surrounding it, creating a blob of light. Software is then used to recognize this blob as a fiducial, located on a grid that is laid over the graphical user interface of the application.

*Rear Diffused Illumination* (RDI) is accomplished by placing an infrared illuminator and an infrared camera below the glass touch surface, and a diffusing projection surface material above. The diffusor material is typically made of mylar or vellum, and allows the projected infrared light to diffuse across its surface while allowing some visible light to pass through as well. When an object comes into contact with the diffusor material it creates a more reflective surface than the area around it, thus reflecting infrared light back towards the camera creating a “blob.”

*Front Diffused Illumination* (FDI) is similar to RDI in that infrared light is projected onto the touch surface. However, as it’s name would suggest the light is projected from above the surface. Again, a diffusor material is placed on top of the touch surface, while an infrared camera is placed below. When an object comes in contact with the diffusor it creates a shadow of infrared light as seen by the camera. The software’s function in the application would be inverted from an RDI installation wherein it would be calibrated to detect absences of light rather than abundances.

*Frustrated Total Internal Reflection* (FTIR) was developed by Jeff Han in 2005. As its name would suggest, FTIR is based on the principle of total internal reflection. Total internal reflection is a phenomenon that occurs when a ray of light passes from air into certain types of materials at an angle larger than a certain angle relative to the normal called the critical angle. When the angle of incidence is greater (closer to parallel with the material) than a particular angle, the critical angle, and the refractive index is lower on the opposite side of the material boundary, the light is internally reflected and travels along, rather passing back through, the boundary, causing total internal reflection.

An FTIR multi-touch setup consists of a frame of infrared LEDs around a glass touch surface. The LEDs shine infrared light inside of the glass plane, causing total internal reflection. When an object (typically, and most effectively, a finger) contacts the plane, it provides a point on the material boundary at which the reflected light becomes “frustrated” and escapes the inside of the plane. This escaping light creates a fiducial, which is then sensed by an infrared camera placed below the touch surface. It should be noted that this type of fiducial is sensed differently than one from RDI because it is a light fiducial against an otherwise dark background, whereas an RDI fiducial is seen as a section of light that is simply brighter than its background.

*Diffused Surface Illumination* (DSI) functions in a similar manner to FTIR in that infrared LEDs are used to shine infrared light inside of the touch surface. However, a specific type of acrylic must be used containing small particles that diffuse the light evenly across the material (such as Acrylite EndLighten). Because the light is being reflected by the boundary of, and also by the particles in, the acrylic, there is simultaneously total internal reflection (light reflected by boundary) and reflection of light through the boundary (light reflected by particles). When an object contacts the touch surface it frustrates the internally reflected light, allowing it escape and create a light fiducial that is brighter than the surround background light (caused by reflection through the boundary). So while the creation of the fiducials is akin to FTIR, the detection of fiducials functions like RDI.

*A Laser Light Plane* (LLP) consists of a plane of infrared lasers directly over a touch surface, and an infrared camera below. This plane is typically achieved by using a grid of lasers. When an object touches the touch surface, it passes through the laser light plane causing the light to reflect through the touch surface to the camera.

*Light Emitting Diode Light Plane* (LedLP), also referred to as an opto matrix or infrared grid, is made up of a touch surface with infrared LEDs projecting infrared light above, with an infrared camera below. Similarly to LLP, when an object comes into contact with the touch surface the infrared light is reflected towards the camera below.

Despite their simple design, laser light planes have been excluded as options for this build. Lasers are considerably more expensive than the materials required for the remaining options, and are also very dangerous. In the interest of avoiding any necessary risk, a laser light plane has been excluded. LED light planes, while similarly simple in their design, have been excluded due to their inaccuracy. Existing LedLP builds require large amounts of infrared light in the plane; a more efficient design can be achieved using one of the remaining technologies.

Frustrated total internal reflection has been excluded as an option for this build. A simple switch of touch surface to construct a diffused surface illumination interface provides better fiducial illumination and detection with an identical number of infrared LEDs in the frame.

Diffused illumination has also been excluded as options as their designs provide less detectable fiducials than diffused surface illumination. Accurate detection of fiducials is critical to the application due to the possibility of extremely high rate and differentiation of object interaction. It should be noted that while this technology works best using fingers to contact the surface due to their size, a silicon mat can be placed on top of the touch surface in order to create a fiducial with greater surface area when the surface is touched or struck by an object with a smaller surface area. This provides the ability for the surface to be “played” using the hands or drum sticks. The silicon will also provide a more realistic feel to the performer.

**Projection**

In all of these installations, a projector can be employed below the touch surface in order to project the graphical user interface of the application onto the surface. In the applications that use a diffusor material, this surface works to provide a projector “screen.” In those that do not use a diffusor material (FTIR, light planes), a projection surface of some kind that passes infrared light can be placed on top of the touch surface.

However, projected images can be distorted easily and are not as bright or sharp as an image displayed by a liquid crystal display (LCD). The use of an LCD would provide a more consistent and user-friendly experience. As such, a brief test was conducted to confirm that infrared light will pass through an LCD, allowing it to be mounted below the touch surface in any of the aforementioned installations without interfering with the multi-touch interface. Further testing showed that infrared light will pass through a standard LCD diffusor unaffected, allowing the use of a standard LCD backlight style, given that the backlight does not produce an infrared light that would interfere with the multi-touch interface. More testing showed that the fluorescent bulbs used for LCD backlighting produced no infrared light interference.

**Multi-touch APIs and Frameworks**

As multi-touch technology becomes more popular the number of open source multi-touch programming interfaces increases. These interfaces have been developed in a number of different programming languages for multiple operating system environments. The following information provides a brief overview of many of these interfaces, noting their benefits and hindrances to the programmer and end user. It should be noted at this point that some of these interfaces are referred to as application programming interfaces (APIs) and others as frameworks. In these cases, an API only provides code which interprets light blobs on a hardware interface and outputs tracking and/or event data (finger press, lift, etc.). It is then up to the client-side application to interpret this data appropriately. A framework is designed to not only recognize and output tracking and event data, but to integrate with the client-side application and provide meaningful input data such as mouse movements, button press events, etc.

The following are tracker-side vision application programming interfaces:

C++ is arguably the most commonly used programming language for multi-touch development. While it is operationally fast, development in C++ takes more time than other more simple, rapid prototype geared languages. Community Core Vision (CCV) is an API that outputs fiducial tracking data and touch events. It is designed to work with multiple multi-touch hardware technologies and to interface with client-side applications via the Tangible User Interface Objects (TUIO) protocol. It is not system dependent.

reacTIVision, originally a C++ framework (now also available for Java and C#), was originally designed to handle physical object and multi-touch finger tracking for the Reactable music synthesizer. It is not system dependent.

TouchLib is another C++ multi-touch API. It only provides touch events, and is only functional in a Microsoft Windows environment.

Cocoa is a Macintosh exclusive API developed by Apple that allows programs to interface with Macintosh OS X and (with the Cocoa Touch extension) iOS. Cocoa applications are typically written in Objective C, but the API can be utilized by Python, Perl, Ruby and other languages and tools. BBTouch and Touché are two examples of Cocoa-based multi-touch vision APIs.

The following are client-side application programming interfaces:

The Bespoke Multi-Touch Framework is written in C#, a language that is exclusively supported by Windows. It provides a Windows mouse emulator, hardware calibration features, and example applications.

The MultiTouch.Framework is, as its name would suggest, a cocoa-based multi-touch framework. It works with any TUIO application, and supports input from many different optical multi-touch interfaces as well as iPhone/iPod Touch and other TUIO-based devices.

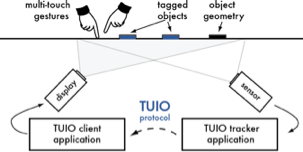
Python is an easy to learn programming language that is quick to develop in, making it the language of choice for programmers who are planning multiple builds throughout a prototyping process. It has its roots in C++, and as such C++ code translates to Python quite well. It is platform independent. PyMT, TouchPy, and 2DCur are good examples of python multi-touch APIs. While PyMT and TouchPy are similar in function, 2DCur is less developed and is also available written for Lily. The 2DCur developers have put most of their effort into developing the Lily library, and as such the Python library is somewhat unorganized and less user friendly.

Grafiti is a C#-based multi-touch framework that is built on the TUIO client. It supports gestural multi-touch interface development. A more developed C# option is Multi-Touch Vista which accepts input from a variety of sources such as TouchLib Vision, a phyiscal mouse, and even Wii remotes to provide standardized multi-touch input to client-side applications.

MT4j is a Java framework. Java, while harder to learn than some other languages, is also completely platform independent. MT4j is designed to support graphics intensive applications, and has most common multi-touch events preprogrammed.

Touchscape SDK is a C++ framework designed for operation in a Windows environment, optimized for Windows 7. It is was conceived with the intention of removing the headaches caused by a multi-oriented multi-touch environment by automatically taking care of orientation, scaling, and other inherent issues.

It is typical in multi-touch programming for the developer to use a TUIO tracker application while developing a client application. A tracker is an application that typically runs in the background and is responsible for receiving user input from a multi-touch interface and providing TUIO output data. A client application is a program that uses TUIO input data, provided by a tracker application, to complete the tasks outlined by the developer. A developer may choose to combine these functions into one single application, but it is not uncommon for developers to provide a client application and expect the user to provide their own tracker. In this case, both programs must be running in order for the client application to be functional. In this project, a TUIO client application is being developed.

Figure I TUIO Client/Tracker Application Diagram

**Glossary**

Sample

Moore’s Law

Synthesizer

Piezo

Velocity

Stroke (different types)

Tip

Shaft

Fiducial

Interface (hardware/physical)

**Appendix D**

**HanDrum**: Requirements Document (version 1.0)

Project: HanDrum

Date(s): 9/28/12

Prepared by: John Cook

Document status: \_X\_ Draft \_\_ Proposed \_\_ Validated \_\_ Approved

**1. Introduction**

This document contains the system requirements for **HanDrum**.

**1.1 Purpose of This Document**

This document is intended to guide development of **HanDrum**. It will go through several stages during the course of the project:

**Draft:** The first version, or draft version, is compiled after requirements have been discovered, recorded, classified, and prioritized.

**Proposed:** The draft document is then proposed as a potential requirements specification for the project. The proposed document should be reviewed by several parties, who may comment on any requirements and any priorities, either to agree, to disagree, or to identify missing requirements. Readers include end-users, developers, project managers, and any other stakeholders. The document may be amended and reproposed several times before moving to the next stage.

**Validated:** Once the various stakeholders have agreed to the requirements in the document, it is considered validated.

**Approved:** The validated document is accepted by representatives of each party of stakeholders as an appropriate statement of requirements for the project. The developers then use the requirements document as a guide to implementation and to check the progress of the project as it develops.

**1.2 How to Use This Document**

We expect that this document will be used by people with different skill sets. This section explains which parts of this document should be reviewed by various types of readers.

**Types of Reader**

Programmers: Sections 2, 3, and 4 are relevant.

Project advisors: All sections are relevant.

End users: Section 3 is relevant.

**Technical Background Required**

An understanding of computer programming, and specifically multi-touch hardware and software is assumed for Sections 3 and 4. Knowledge of the performance of percussion and other instruments is assumed for Section 2.

**Overview Sections**

Sections 1.3, 2, and 3.1 provide an overview for the application that may be understood without any technical knowledge. These sections may also be read prior to the others in order to provide an understanding of the scope and intended functions and uses of the application.

**Reader-Specific Sections**

As this project has been designed with the end user in mind, Section 2 has been written specifically to satisfy the interests of, and explain the application to the user.

**1.3 Scope of the Product**

**HanDrum** is intended to be a software component of an electronic hand drum. This instrument will respond to the performer in the same manner as its corresponding acoustic instrument. While this program will not provide an interface for such interaction with all types of musical instruments, such an interface has yet to be developed for *any* type of instrument. This application is a study in the way performers interact with electronic instruments, and is intended to be the first of a number of projects that attempt to exactly replicate the physical interaction of with acoustic instruments in their electronic counterpart.

**1.4 Business Case for the Product**

As the first application of its kind, **HanDrum** holds a particularly opportune market position. It is the intention of the creator to integrate this application with multiple types of hardware in order to maximize its marketability.

**1.5 Overview of the Requirements Document**

Multi-touch interface, TUIO compatible

Must respond to user input identically to a physical drum. Therefore, it will:

Alter output sound based on method of input (strike on multi-touch surface)

Alter output volume based on velocity of input (strike on multi-touch surface)

**2. General Description**

This section will give the reader an overview of the project, including why it was conceived, what it will do when complete, and the types of people we expect will use it. We also list constraints that were faced during development and assumptions we made about how we would proceed.

**2.1 Product Perspective**

**HanDrum** fills a gap in the functionality of electronic percussion instruments, and in the method of design for electronic musical instruments. This instrument will provide the benefits of an electronic instrument along with the “playability” of an acoustic instrument; no learning curve will be present with **HanDrum**. The benefit scope of the application will be two fold. First, percussionists will directly benefit by having it available for their performance. Second, it is the hope of the developer that this approach to electronic instrument design will spark development on other types of similarly playable electronic instruments. This would be of benefit to the music performance community as a whole. The application is being developed by its sole stakeholder, John Cook.

**2.2 Product Functions**

**HanDrum** will perform exactly as an acoustic hand drum. It will output various drum sounds, depending on the way the multi-touch surface is struck.

**2.3 User Characteristics**

**HanDrum** is intended to be used by percussion performers. As such, the user will have a musical ability to include proper hand percussion technique. It is inherent in the perspective of this application that no technical background will be necessary to use it for its intended purpose.

**2.4 General Constraints**

This software must be developed in order to interact with existing multi-touch hardware and interfacing software.

**3. Specific Requirements**

This section of the document lists specific requirements for **HanDrum**. Requirements are divided into the following sections:

Must respond to user input identically to a physical drum. Therefore, it will:

Alter output sound based on method of input (strike on multi-touch surface)

Alter output volume based on velocity of input (strike on multi-touch surface)

The application has no reporting requirements at the time of this writing.

Must support standard TUIO data input, either by collecting and interpreting the data or interpreting after receiving it from a TUIO interpreting wrapper application.

No interface requirements specified at this time. May be a mockup of a fiber drum head that “vibrates” when struck.

**3.1 User Requirements**

Knowledge of proper hand drum technique.

**3.2 Reporting Requirements**

The application has no reporting requirements at the time of this writing.

**3.3 System and Integration Requirements**

Supports TUIO framework

Extremely fast response time

Will run as .app on Macintosh OS X

**3.5 User Interface Requirements**TUIO compatible multi-touch hardware interface. Independent TUIO wrapper must be functioning in order to interpret user input data for **HanDrum**.

**4. High-Level Technology Architecture**

Python; must be able to be translated into C++ for future distribution to other types of hardware.

**Glossary**

Multi-touch - refers to the ability of a touch sensing user interface to recognize the presence of two or more points of contact with its surface.

Percussion - a musical instrument that is sounded by means of striking, scraping, shaking, or plucking.

TUIO - tangible user interface objects

Wrapper application - a program or script that makes it possible for another program to run