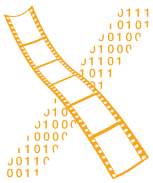

Chair for Computer Science 10 (Media Computing and Human-Computer Interaction)

RWTH AACHEN UNIVERSITY

Overcoming Latency Challenges of Tangible Interactions on macOS

Bachelor's Thesis
submitted to the
Media Computing Group
Prof. Dr. Jan Borchers
Computer Science Department
RWTH Aachen University

by
Jovan Medianto Riman

Thesis advisor:
Prof. Dr. Jan Borchers

Second examiner:
Prof. Dr. Ulrik Schroeder

Registration date: 26.07.2023
Submission date: 11.10.2023

Zentrales Prüfungsamt/Central Examination Office

Eidesstattliche Versicherung
Statutory Declaration in Lieu of an Oath

Name, Vorname/Last Name, First Name

Matrikelnummer (freiwillige Angabe)
Matriculation No. (optional)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/ Masterarbeit* mit dem Titel
I hereby declare in lieu of an oath that I have completed the present paper/Bachelor thesis/Master thesis* entitled

selbstständig und ohne unzulässige fremde Hilfe (insbes. akademisches Ghostwriting) erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.
independently and without illegitimate assistance from third parties (such as academic ghostwriters). I have used no other than the specified sources and aids. In case that the thesis is additionally submitted in an electronic format, I declare that the written and electronic versions are fully identical. The thesis has not been submitted to any examination body in this, or similar, form.

Ort, Datum/City, Date

Unterschrift/Signature

*Nichtzutreffendes bitte streichen

*Please delete as appropriate

Belehrung:
Official Notification:

§ 156 StGB: Falsche Versicherung an Eides Statt
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.
Para. 156 StGB (German Criminal Code): False Statutory Declarations
Whoever before a public authority competent to administer statutory declarations falsely makes such a declaration or falsely testifies while referring to such a declaration shall be liable to imprisonment not exceeding three years or a fine.
§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt
(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.
Para. 161 StGB (German Criminal Code): False Statutory Declarations Due to Negligence
(1) If a person commits one of the offences listed in sections 154 through 156 negligently the penalty shall be imprisonment not exceeding one year or a fine.
(2) The offender shall be exempt from liability if he or she corrects their false testimony in time. The provisions of section 158 (2) and (3) shall apply accordingly.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:
I have read and understood the above official notification:

Ort, Datum/City, Date

Unterschrift/Signature

Contents

List of Figures

List of Tables

Abstract

Tangibles are physical objects that enable users to interact with digital contents. They provide haptic feedback and provides users with new ways to interact with a device. To allow tangible interactions in macOS, MultiTouchKit (MTK) was developed [Linden, 2015]. However, this framework has poor performance in its touch processing, causing a huge amount of latency. This leads to poor experience with tangible interactions using this framework.

This bachelor's thesis aims to optimize latency in MTK to improve tangible interaction in macOS. We optimized the latency by modifying the way MTK receives touch input by receiving them via USB and how traces are visualized in the MTK. Our results shows an average latency of 94.97 ms with no trace visualization and 102.23 ms with the new trace visualization. Furthermore, our results also shows that the newly modified trace visualization had less latency compared to the old trace visualization.

Acknowledgements

I would like to thank Prof. Jan Borchers for supervising my thesis and Prof. Ulrik Schroeder for being my second examiner.

I would also like to thank Adrian Wagner and Sebastian Hueber for being my advisors for this thesis and for providing me feedback and guidance throughout the thesis.

Lastly, I would like to thank my family and friends who supported me during my studies.

Conventions

Throughout this thesis we use the following conventions.

Text conventions

Definitions of technical terms or short excursus are set off in coloured boxes.

EXCURSUS:
Excursus are detailed discussions of a particular point in a book, usually in an appendix, or digressions in a written text.

Definition:
Excursus

Source code and implementation symbols are written in typewriter-style text.

```
myClass
```

The whole thesis is written in American English.

Chapter 1

Introduction

LATENCY/LAG:
The time difference between user input and output response of a system [MacKenzie and Ware, 1993].

Definition: *Latency/Lag*

Latency or lag is known to impact the usability of devices [MacKenzie and Ware, 1993]. A higher latency can impact user performance and make systems feel less responsive and interactive. Therefore, we need to consider how latency affects system interactions and to optimize latency as much as possible in a system.

Latency impacts usability of devices

Tangibles are physical objects that enable users to interact with digital contents by coupling together virtual objects with physical objects [Ishii, 2008]. Tangibles provide users with haptic feedback, which can increase precision when manipulating digital objects [Fitzmaurice and Buxton, 1997] and allows for eyes-free interaction [Weiss et al., 2009].

Tangibles provide new input and output modalities

To enable tangible interactions on macOS, a framework called Multitouchkit (MTK) was developed [Linden, 2015]. This framework allows applications to detect tangibles on a touch display. However, MTK has a poor performance in detecting and processing touches, leading to a huge amount of latency. This leads to a poor user experience when interacting with tangibles.

MTK has poor performance, leading to huge amounts of latency

We decided to optimize latency in MTK

This thesis aims to improve tangible interactions in macOS by optimizing latency in MTK. To do this, we modified two parts of the touch processing that we believe to impact latency the most: how MTK received touch inputs and how visual feedback is given.

Thesis structure

This thesis has the following structure. In the second chapter, we will take a look at some related work on how humans use the sense of touch in their everyday life, the advantages and challenges tangible interactions provide, how latency is measured and the impacts of that latency have on humans. In the third chapter, we will explore how MTK processes touch input and the sources of latency that we discovered in the touch processing. We will show how we optimized latency in the MTK. In the fourth chapter, we will measure the latency of the optimized framework and discuss the results. In the final chapter, we conclude with a summary and possible extensions of the thesis.

Chapter 2

Related Work

2.1 Touch and Somesthesis

Similar to all senses, touch is one of the most fundamental means of contact with the external world [Jenkins and Lumpkin, 2017]. Touch allows us to feel textures, pain, heat and others [Field, 2003]. However, unlike other senses, touch is intimate as it requires direct contact with the skin.

Sense of touch

Touch is one of our primary ways to communicate non-verbally [Gallace and Spence, 2010]. We use touch to socially convey emotions and is essential for our physical and emotional wellbeing. Despite its importance in communication, the study of this sensory system as a communication channel has been given little attention compared to facial and vocal channels [Stack, 2007].

Touch is used in non-verbal communication

The sense of touch is part of the somesthetic sense. This sense includes not only cutaneous senses (touch) but also includes kinesthesia, the ability to sense the movement and position of our limbs [Craig and Rollman, 1999]. In haptics, the somesthetic sense is being stimulated through force-feedback, which is dependant on the the person's limb movement.

Somesthetic sense

Somesthesis plays an important role in interacting with objects in a real or virtual environment [Robles-De-La-Torre,

Somesthesis is important in system interaction

2006]. Information received through the somesthetic sense is crucial for fast and accurate interaction with the environment. Without this, achieving top performance in tasks that requires a high level of dexterity is extremely difficult. The lack of somesthetic capabilities in a person can cause issues such as increase in the difficulty in performing tasks that involves cognitive and fine motor skills, a major loss of precision and movement speed, and major increase in the difficulty of learning motor skills.

Lack of somesthetic feedback can reduce user performance

A lack of somesthetic feedback can lead to a decrease in user performance [Robles-De-La-Torre, 2006]. User performance could worsen if additional sensory information is lacking. However, there still remains the question to identify which somesthetic information is required for different tasks. As an example, people type quicker on a physical keyboard compared to a touch display keyboard due to the haptic feedback that the physical keyboard provides [Sears, 1991]. Overall, somesthetic feedback is something that needs to be considered in system interactions.

2.2 Tangible Interactions

GUIs are mostly used today

Most interactions with the digital world involves Graphical User Interfaces (GUIs). GUIs represent information graphically on the screen through pixels [Jansen, 1998]. We interact and manipulate graphical components in the GUI by using controllers such as the mouse and keyboard. Although GUIs provide the capability to emulate various medias graphically, our interaction with GUI screens are inconsistent with our interactions with the physical environment within which we live [Ishii, 2008].

Tangibles provide new input and output modalities

Tangible interaction allows user to interact with digital contents on a touch display by manipulating physical objects by coupling virtual objects with physical objects[Schneider et al., 2011, Fitzmaurice et al., 1995]. They take advantage of the human's capability to grasp and manipulate physical objects by providing haptic feedback and they also provide new ways for users to interact with digital content. An example can be seen in figure 2.1.

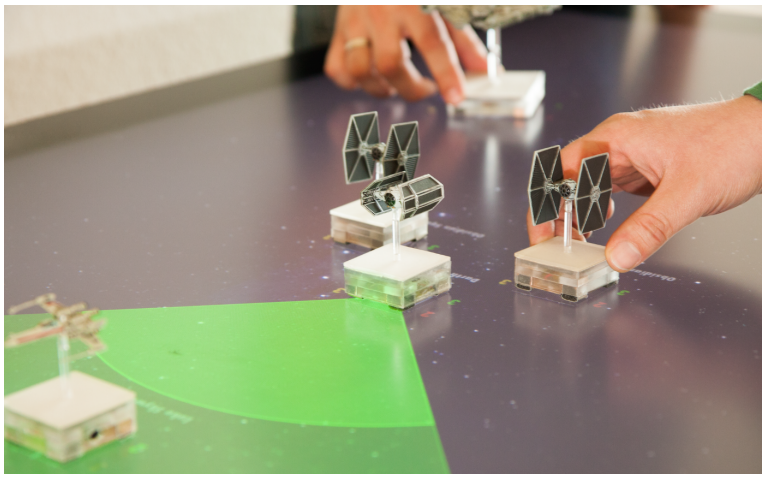

Figure 2.1: Example of tangible interaction from Voelker et al. [2015a]. This image shows an interactive board game that uses tangibles to represent Star Wars ships on a capacitive touch display.

We will go through some related work to explore the advantages that tangible interactions provide. Furthermore, we will explore some of the challenges in creating a tangible user interface.

2.2.1 Advantages of Tangible Interactions

Here we present four advantages of tangible interactions.

Tangibles are intuitive to use

First, tangibles are intuitive to use because of our prehensile behavior [Fitzmaurice et al., 1995]. Due to the physicality of tangibles, they provide a richer affordance compared to virtual objects [Schneider et al., 2011]. These affordances gives a strong feeling of directness to the tangibles, making them intuitive to use. Furthermore, due to the tight coupling between tangibles and virtual objects, tangibles allow us to take advantage of our innate spatial and environment skills and our ability to physically manipulate objects [De Raffaele et al., 2018].

Tangibles increase user performance

Second, tangibles increase user performance when per-

forming tasks. A study done by Fitzmaurice and Buxton [1997] showed an increase in precision when operating tangibles. Additionally, a study done by Voelker et al. [2015b] showed that users were on average faster and less error-prone when using tangible objects then virtual objects on a touch display.

Tangibles allow eyes-free interaction

Third, tangibles allow for eyes-free interaction [Weiss et al., 2009]. Due to the haptic feedback they provide, users are able to apply their visual focus elsewhere while operating tangibles. The experiment by Voelker et al. [2015b] showed similar user performance when operating tangibles eyes-free compared to eyes-on.

Tangibles enhance collaborative activities

Fourth, tangibles enhances collaborative activities [Schneider et al., 2011, Fitzmaurice et al., 1995]. They provide new forms of collaborative interactions that GUIs do not provide. Cognitive load is also shown to decrease during collaborative activities with tangible interactions [Schneider et al., 2011]. Furthermore, tangibles are shown to increase awareness of the action of other users [Cherek et al., 2018]. The study showed that this was possible due to physicality of the tangible which provides cues through the reflective properties of the surface, the potential sound the tangibles create, and awareness of the movements of other users.

2.2.2 Tangible User Interface Challenges

Tangible User Interface

Tangible User Interface (TUI) is an interface that enables tangible interaction. Creating such interface has its own challenges. Here, we will explore some related work and present three challenges in creating a tangible user interface.

Detecting tangibles is challenging

The first challenge is the process of detecting tangibles [Klemmer and Landay, 2004]. There are multiple methods that were proposed and suggested.

Detect tangibles via touch display

A method to do this is to detect tangibles via a touch display. Passive Untouched Capacitive Widgets (PUCs) (see figure 2.2) are tangibles that can be detected by unmodi-

Figure 2.2: Passive Untouched Capacitive Widgets (PUCs). They are uniquely identified through the pattern of the contact points. [Voelker et al., 2013]

fied capacitive touch displays [Voelker et al., 2013]. To detect them, the interface must be able to recognize a specific pattern formed by the contact points of the PUCs. Furthermore, PUCs have a problem where they disappear after 5 – 30 seconds due to the filtering mechanism of the touch display [Voelker et al., 2015a]. This makes it difficult to determine whether a tangible was filtered out or whether it was intentionally removed from the touch display. This needs to be handled by the TUI.

Persistently Trackable Tangibles on Capacitive Multi-Touch Displays (PERCs) (see figure 2.3) was introduced by Voelker et al. [2015a] to solve the issue PUCs have. PERCs has the capability to detect whether it is on a capacitive touch display or not. To detect PERCs, the TUI must be able to communicate with the PERCs to determine whether the PERCs is on the capacitive touch display.

Detect tangibles using computer vision

Klemmer and Landay [2004] suggested using computer vision as a technique to detect tangibles. This method can be characterized as being either the *tag* variety or the *artificial intelligence (AI)* variety. The *tag* variety uses computer vision to determine whether an object has a specific property that's predetermined. For example, if an object has a specific qr code on it, it will be classified as a specific tangible. This variety is deemed to be more robust and computationally cheaper. However, tagging objects can take a large amount of time and this property must be visible. The

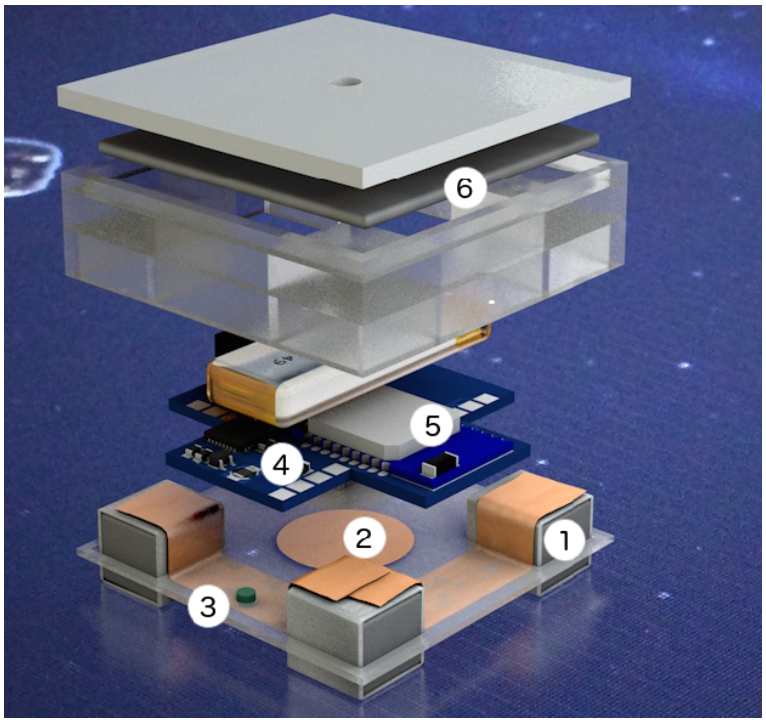

Figure 2.3: Components of Persistently Trackable Tangibles on Capacitive Multi-Touch Displays (PERCs). 1) marker pattern, 2) field sensor, 3) light sensor, 4) micro controller, 5) BLE module, and 6) a lead plate. [Voelker et al., 2015a]

AI variety observes the properties of the object, such as its color, shape and size, and classifies them informally in a group based on those properties. However, this method is deemed to be computationally expensive and not very robust.

Abstracting tangibles is challenging

The second challenges is the process of abstracting tangibles [Klemmer and Landay, 2004]. PUCs are uniquely determined by the geometric patterns of the contact points [Voelker et al., 2013]. This limits the number of uniquely identifiable tangibles due to the limited number of contact point patterns that can be accommodated [Voelker et al., 2015a]. There is also the potential error where a tangible is identified due to active touches having the same contact point pattern to a tangible. PERCs do not have this issue because every PERC contains its own Bluetooth mod-

ule, making them uniquely identifiable. However, as mentioned earlier, the TUI must be able to communicate with the PERCs. For the computer vision method, is it unknown whether they could be uniquely identified.

Tangible interaction have strict timely requirements

The third challenge involves addressing parallel and continuous interaction [Shaer and Hornecker, 2009]. Due to the haptic properties of tangible interaction, a strict timely requirements is required for seamless operation [Schneider et al., 2011]. Furthermore, a noticeable latency can disrupt the users perceived coupling between virtual objects and its corresponding physical object [Fitzmaurice et al., 1995]. Overall, latency needs to be minimized in tangible interaction.

2.3 Latency

Latency is unavoidable

Latency is an unavoidable part of any system [MacKenzie and Ware, 1993]. This is caused by a combination of the input device, software and the output device.

To further understand the importance of optimizing latency, we will go through some related work and explore some methods to measure end-to-end latency for touch interactions, the impacts of latency on user interactions and how well humans are able to perceive them.

2.3.1 Measuring Latency

Measure latency with high speed camera

Kaaresoja and Brewster [2010] introduced a method to measure latency by using a high speed camera to capture a high frame rate video to detect visual feedback. The method uses a make-up mirror to see the touch occurrence from a side-view (see figure 2.4). This solution is simple to implement and can be implemented on both capacitive and resistive touch displays. However, this method requires a manual analysis of to measure latency which is time consuming and can cause potential errors.

Figure 2.4: Hardware to measure latency by Kaaresoja and Brewster [2010]. A touch display is placed on the table along with a mirror beside it. A camera is mounted to capture a slow-motion video with both the mirror and the touch display in the frame.

Other suggestions

The study also provided other suggestions. A programmable robot arm with a force gauge was initially suggested to measure when touch occured. However, this method was too expensive to implement, unportable, and challenging for a capacitive touchscreen to detect the touch. An alternative approach was to build a stylus-like device with a sensitive switch on the tip and an LED on the other end to measure when touch occured. With this approach, the LED would light up as soon as the stylus touches the display. However, this approach requires building new hardware and was challenging for a capacitive touchscreen to detect the touch.

Measure latency using vibration sensor and photo-diode

Casiez et al. [2017] presented another method to measure end-to-end latency for a touch display or for a physical button. For this method, an Arduino Leonardo[1], a vibration sensor and a photo-diode is used (see figure 2.5). A vibration sensor is used to detect when a touch or button click occurred and is attached to the user's finger, mimicking the interaction. The photo-diode is used to detect when the screen responds and is attached to a part of the touch

[1] https://docs.arduino.cc/hardware/leonardo

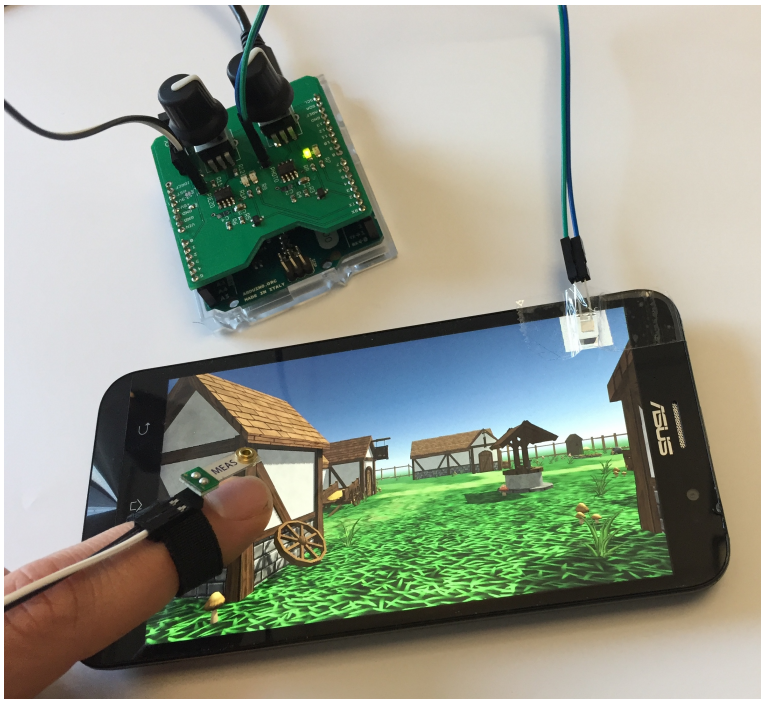

Figure 2.5: Hardware to measure latency by Casiez et al. [2017]. The vibration sensor is attached to the users finger and is connected to the Arduino board. The photo-diode is attached to the touch display and is connected to the Arduino board

display, which flashes upon detecting touch input. The latency measured is the time difference between the time the vibration sensor reacts and when the photo-diode reacts. This solution is automated and cheap to implement.

Measure latency using dragging tasks

Bérard and Blanch [2013] proposed two methods to measure latency in touch-enabled systems. For both methods, latency is estimated by measuring the gap between the the leading trace on the display and the current position of the touch. Automated data processing is included for both methods. The first method is called the High Accuracy Approach which provides a large sample set of accurate latency measurements but requires an external camera and careful calibration. The second approach is called the Low Overhead approach which does not offer much accuracy compared to the former approach (latency differs no fur-

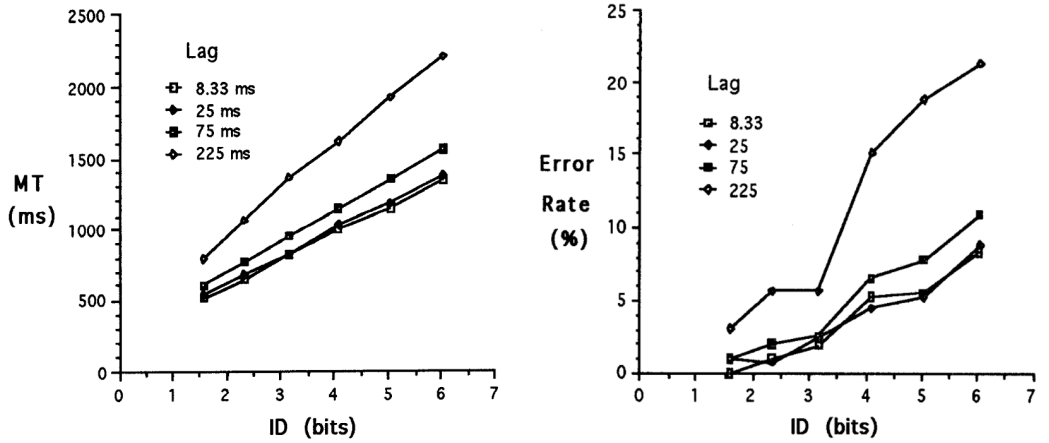

Figure 2.6: Graph from MacKenzie and Ware [1993] showing an increase in both movement time (MT) (left) and error rate (right) in relation to the index of difficulty (ID) over 4 levels of lag.

ther than 4 ms) but is simpler to implement.

2.3.2 Impact of Latency

Lag decreases user performance

Lag can negatively impact motor-sensory tasks on interactive systems. MacKenzie and Ware [1993] conducted an experiment to measure and model the effect of lag on speed, accuracy, and bandwidth of human motor-sensory performance in interactive tasks. The study showed that a higher latency increased both movement time and error rate (see figure 2.6). The effect worsens as the difficulty of task increases. The study concluded that the effect of 75 ms lag can easily be measured, and a 225 ms lag caused a significant decrease in performance.

Latency can impact brain activity

Latency can negatively impact the brain activity of users. A study done by Somei et al. [2023] measured the brain activity to investigate the effects of latency on brain activity. The study concluded that a latency of 150 ms leads to a reduction in the operator's sense of urgency – the feeling that the operator is the one controlling their actions. The effect worsens as latency increases. Additionally, an increase in latency causes users to feel more stressed when operating the device. However, the perception of stress and brain

activity differs depending on how well latency was recognized.

Lag has negative impact in real-life applications

Lag is shown to have a negative impact in real-life applications. In a gaming context, lag can cause a game to feel unresponsive, leading to a reduction in player performance and their quality of experience [Liu and Claypool, 2022]. In a surgical setting, performing remote surgery with high latency can be dangerous [Rayman et al., 2005].

2.3.3 Perception of latency

The negative impact mentioned earlier only occurred when the lag was perceived in the first place. Here we see how well latency is perceived in touch-enabled systems.

Lag is easily perceived in direct touch systems

To analyse user's perception in touch-based systems, Deber et al. [2015] conducted an experiment to measure latency perception for direct and indirect input systems for both tapping and dragging tasks. Indirect input systems separates the input device and the output device while direct input systems allows users to provide input directly on the touch display. In this experiment, users use their fingers to perform the corresponding tasks. The first experiment concluded that 11 ms for direct touch and 55 ms for indirect touch were the thresholds for perceiving latency for dragging tasks. For tapping tasks, 69 ms for direct touch and 96 ms for indirect touch were the thresholds. These results indicate that it was easier to perceive lag in direct touch systems compared to indirect touch systems.

Small improvements in latency are noticeable

In the same study by Deber et al. [2015], a second experiment was conducted to investigate how well people perceive small improvements in latency in both direct and indirect input systems for both tapping and dragging tasks. It was shown that improvements in latency as small as 8.3 ms are noticeable, indicating that small improvements in latency are noticeable in touch-enabled systems.

Latency impacts tangible interactions

Ng et al. [2014] conducted an experiment to measure latency perception for dragging and scribbling tasks for sty-

luses, which can be considered a pen-shaped tangible. They observed that very low levels of latency can be perceived for both dragging tasks (2 ms – 7 ms) and scribbling tasks (7 ms – 40 ms). This indicates that this noticeable small improvement in latency does not only apply to touches conducted by fingers, but applies to tangibles as well.

Chapter 3

Latency Challenges in Tangible Interactions

Overview of this chapter

In this chapter, we will first explore how MTK processes touch input. Next, we will go through some sources of latency that was discovered during the touch processing and explain how they caused a large amount of lag. Finally, we will present how latency was optimize and a problem that we faced doing this.

3.1 Touch processing

This section is a summary of the touch processing of MTK from Linden [2015]. This allows us to understand how MTK receives touch input. Here, we define a touch as anything that is recognized as a touch by the touch display. This could be a human finger or a tangible.

MTK uses SpriteKit

MTK uses Spritekit[1] to render graphics. SpriteKit uses a rendering loop (see figure 3.1) to process and to render contents of a frame. Touch processing occurs in the start of every frame, specifically when the `update:` method is called. This method is called on the active SKScene, which we will

[1] https://developer.apple.com/spritekit/

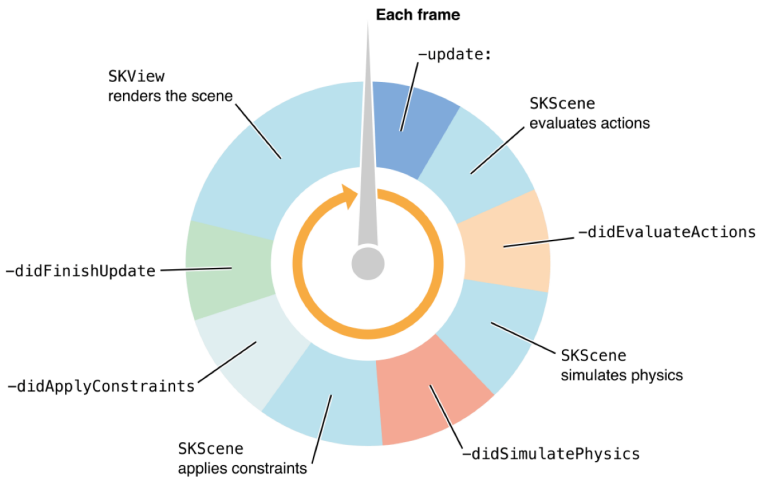

Figure 3.1: Rendering loop of SpriteKit. This image was taken directly from Apple's official documentation.

call a *scene*. SKScene is a subclass of SKNode, which we will call a *node*.

Nodes in SpriteKit

In SpriteKit, nodes are the building blocks for the content in the scene. The scene is the root node and contents on the screen gets rendered based on the tree of node objects.

We focused on the initialization of touch processing

MTK splits the touch process into two parts: the initialization and the recursive touch processing. We will focus on the initialization as this is where touch input is being received and processed. It consists of the following steps:

1. Update all MTKTraces
2. Call `preprocess:` of global delegate
3. Update cursors
4. Update tangibles
5. Update global gesture recognizer
6. Distribute traces to SKNodes in scene
7. Call `postProcess:` of global delegate
8. Start recursive scene processing

Traces are updated

In the first step, MTK updates all MTKTraces. An MTKTrace, which we will call a *trace*, represents the lifetime of a single touch. A trace contains information such as its identifier and position and is saved every frame while the touch was recognized by the touch display. Traces allows MTK to interpret touch inputs. This step involves creating and updating MTKTraces.

Traces are created from their input source

Before the modification, MTK supports two types of input sources: `MouseInputSource` and `JSONInputSource`. `MouseInputSource` is responsible for converting mouse events into traces. `JSONInputSource` is responsible for interpreting information from a JSON object and converting them into traces. This is the main input source that allows MTK to receive touch input. The end result is an array of active traces from all sources. The result is used for the remaining steps.

Preprocessing of traces

In the second step, `preprocess:` of the global delegate is being called. This step allows developers to manipulate traces before the actual processing begins.

Cursors are updated

In the third step, cursors are updated. Here, visual feedback is added to the scene to notify users of the current position of the active traces in the scene. This is mainly used during debugging and can be disabled during normal use.

Two types of visual feedback for traces

There are two ways MTK provides visual feedback for traces: cursor visualization and trace visualization. Cursor visualization provides visual feedback on the current position of the active traces. Trace visualization provides visual feedback of the current position as well as the past positions of the active trace. The number of past positions is determined by the buffer size, which can be configured in the source code.

Tangibles are updated

In the fourth step, we update tangibles. This step takes all the traces and interprets them to recognize tangibles that were initially registered to the MTK. As of now, MTK supports PUCs [Voelker et al., 2013] and PERCs [Voelker et al., 2015a]. It also deals with recovery of tangibles in the case that the touch points of the tangible disappears for a short amount of time and provides user with visual feedback for

the tangibles that are active in the scene.

Gestures are detected

In the fifth step, the global gesture recognizer gets updates. The gesture recognizer looks for certain patterns in the path of the traces and calls a method based on it.

Trace distribution to nodes

In the sixth step, trace distribution to the nodes occurs in the scene. Traces that are not recognized as gestures or tangibles need to be bound to one of the nodes in the scene. This step deals with distributing traces to the correct nodes.

Post-processing of traces

In the seventh step, `postprocess:` of the global delegate is called. This step allows developers to manipulate traces after traces has been processed.

Recursive touch processing starts

All these steps are performed at the beginning of each frame. After the initialization finishes, the recursive touch process starts which is a function call of the active scene and is called recursively by the children of the scene and their children.

3.2 Sources of latency

After understanding the basics on how MTK receives touch input, we will now go over some of the latency challenges that we face with this framework and the reason they occur.

Touch processing is slow

One of the challenges we face is that receiving touch input is slow. Obtaining traces from touch input was done through the `JSONInputSource`. This input source allows the MTK to receive touch input data from JSON objects which is sent via network. This allows for flexibility as this input source can work with any hardware that can send input data via network. However, this input source provides a huge latency due to the complex process that occurs before MTK receives the touch input (see figure 3.2).

macOS has no native support for external touch display

To receive touch input, the touch display must first be able to send it's data to the MTK. However, macOS does not have native support for receiving touch input from external touch displays. `JSONInputSource` was created as an

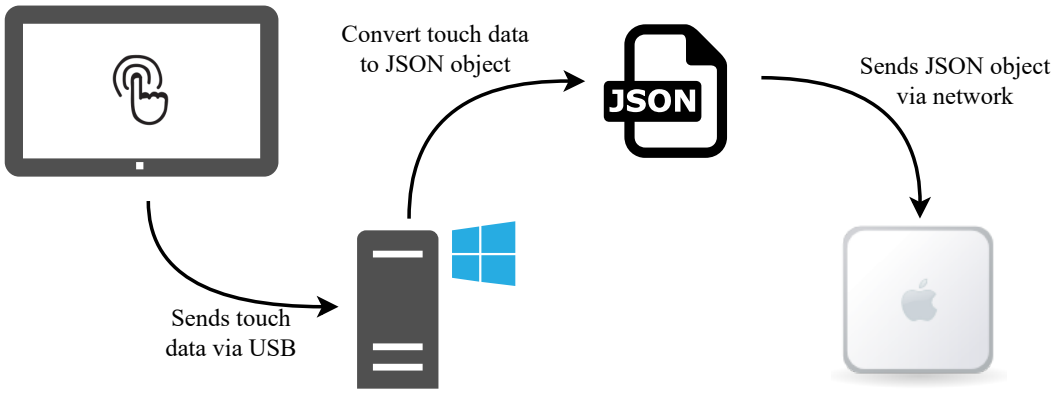

Figure 3.2: Touch processing using the `JSONInputSource`.

alternative method to solve this issue.

Complex setup is needed

A setup is required before this input source can be used. First, a Windows computer is required to read touch input from the touch display. The Windows computer requires a client to read the touch data from the touch display, convert the data to a JSON object, and send the JSON object to a specified IP address and port. The IP address and port also needs to be configured in the MTK to receive JSON objects from the Windows computer. Finally, the touch display is connected to the Windows computer via USB and MTK is ready to receive touch input from `JSONInputSource`.

Complex process to receive touch input

Upon receiving the touch input, the client on the Windows computer reads the touch data through the USB and converts the data to a JSON object. This JSON object is formatted in a specific way, which is specified in the MTK. After the conversion, the client sends the JSON object to the MTK with the specified IP address and port. After receiving the packet via network, MTK unpacks the packets and interprets the touch input from the JSON object. After this, MTK finally receives a set of touch inputs.

This method of receiving touch input requires a lot of converting and abstracting of data, which leads to a huge amount of latency. Furthermore, this input source brings in complexity to the setup process.

Trace visualization is heavy

Another challenge is the heavy processing required to visu-

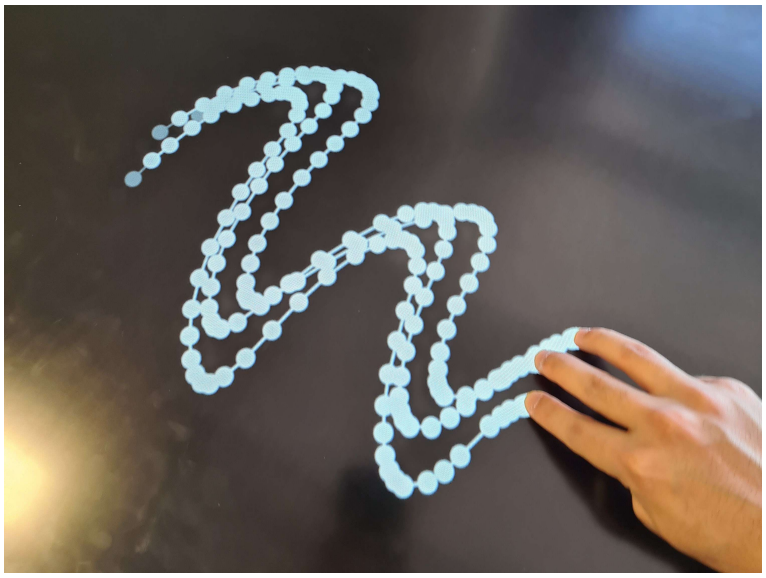

Figure 3.3: Trace visualization in MTK. This is mostly used during debugging to allow developers to see the history of traces and its path. White circles represents past location of traces.

alize traces. To provide visual feedback to the user, nodes for trace visualization are added to the scene. These nodes come in the form of circles and lines (see figure 3.3). However, these nodes have no reference to its trace. This means that MTK does not know which trace visualization nodes to delete when a trace disappears. To allow for this, all nodes for trace visualization are deleted at the start of every frame. If a trace remains active, temporary nodes are created again for all past locations inside the buffer. This process repeats every frame.

This method is simple implement but caused a huge amount of latency due to the overhead in deleting and adding nodes to the scene every frame. Although trace visualization is primarily used for debugging, improving latency here will lead to a better debugging experience.

Hardware limitation

Another challenge is hardware limitation. One hardware that can cause additional latency is the touch display, specifically the refresh rate of the display. The refresh rate

of a display shows how frequently the display updates the screen. A higher refresh rate allows the touch processing loop to occur more frequently as well as allowing the display to update more often. It was also shown that doubling the refresh rate can reduce latency by up to 15.5 ms [Casiez et al., 2017]. Another hardware that can impact latency is the processor. A fast processor will be able to process things more quickly and efficiently, leading to less latency. Unfortunately, hardware limitations cannot be compensated through software.

3.3 Solution

In this section, we will explain our implementation to optimize latency in MTK. We did this by modifying how touch input is received and how trace visualization works.

3.3.1 Receiving Touch Input via USB

Receive touch input via USB

To omit the complex process of the touch processing done by `JSONInputSource`, we decided to modifiy MTK to receive touch input data directly through USB. We added a new input source called `TouchInputSource`, which will be responsible for receiving touch input via USB and converting them to traces. To do this, a framework called Touch Up was integrated into the MTK.

Definition: *Human Interface Device*

HUMAN INTERFACE DEVICE:
A user interface for types of computer device that are operated by humans. Commonly referred to as "HID". This definition is taken from the Device Class Definition for HID 1.11.

Touch Up framework

Touch Up[2] is a framework that provides access to active touches that is recognized by the touch display. Touch Up uses the IOHIDManager[3] API collection. This API is pro-

[2] https://hci.rwth-aachen.de/touchup
[3] https://developer.apple.com/documentation/iokit/iohidmanager_h

vided by Apple and provides tools for developing drivers for human interface devices. IOHIDManager allows Touch Up to detect touch input by accessing the HID interface of the touch display, which acts as a bridge for the communication between macOS and the touch display. This allows macOS to receive data packets directly from the touch display via USB.

Windows Touchscreen Collection

Most touch displays follows the Windows Touchscreen Collection[4] protocol. This protocol is used for touchscreen reporting in Windows 10 and later operating system. Touch Up uses IOHIDManager to receive data packets via USB and interprets the data based on the protocol.

Receiving touch input via USB

Before touch processing begins, MTK starts the `TouchInputManager`. This is imported from Touch Up. `TouchInputManager` is responsible for receiving touch input from the touch display and interpreting the data based on the Windows Touchscreen Collection protocol. Once the touch processing begins, `TouchInputManager` reads data coming in from the touch display via USB every frame, interprets them and stores the touch data in a set in the MTK. The set contains the contact ID and a normalized global coordinate of every active touch in that frame.

Touch input gets converted into traces

To obtain traces from the set of touch inputs received, MTK first converts the coordinate of every touch into a pixel-based global coordinate in the macOS coordinate system. This is due to the different origin points of the coordinate system in macOS, which is in the bottom-left corner, and in Touch Up, which is in the top-left corner. For every touch input, the pixel-based global coordinate is then converted into a pixel-based local coordinate of the scene. This allows MTK to check if all active touches are inside the scene. Touches that are outside of the scene will not have a trace created for it. Finally, MTK receives a set of traces that are active in the scene.

[4] https://learn.microsoft.com/en-us/windows-hardware/design/component-guidelines/touchscreen-required-hid-top-level-collections

Advantage of using USB

Although `JSONInputSource` provides us with flexibility due to its compatibility with all input sources, there are a couple of advantages that receiving touch input using the new source provides over the old input source.

Less processes required

First, the new input soruce allows MTK to receive touch input directly from the touch display via USB. This method requires less processes than `JSONInputSource`, leading to a reduction in latency.

Setup is easier

Second, receiving touch input via USB dismisses the complex setup required when using `JSONInputSource`. Because USB is designed to be compatible with all computers, getting touch input from a touch display with the MTK is as simple as plug and play.

Problems with using USB

MTK cannot detect all touches on Surface Hub

Despite all the latency advantages that receiving touch input via USB provides, there exists one problem. MTK is not capable of detecting all touches from certain touch displays. We will explain this problem with an example involving the Surface Hub.

Surface Hub has two modes to receive touch input

The Surface Hub has two modes that allows it to send touch input to an external device: Projection Mode and Guest Mode. Projection mode is activated when the operating system of the Surface Hub detects an external device that is connected to the Surface Hub. Guest Mode is activated by manually switching the display input source to the source that is connected to the computer. The difference is that the user interface of the Surface Hub is still active in Projection mode while it is inactive in Guest Mode.

We focused on Guest Mode

The Microsoft Surface Hub can support up to 100 touches simultaneously[5]. From our analysis, this only occurs in

[5]https://support.microsoft.com/en-us/surface/surface-hub-tech-spec-4b57f72c-dc1c-28d7-959f-3d95eda7708f

Guest Mode. In Projection Mode, the Surface Hub can have a maximum of 10 active touches. We focused on using the Guest Mode to take advantage of the maximum number of touches that the Surface Hub supports.

Touch Up can only detect 5 touches

Touch Up is only able to detect a maximum of 5 touches despite the maximum number of contact points that the Surface Hub supports. To further understand this, we analysed the data that is being sent and received through the HID interface on a Windows computer. We analysed the HID report descriptor, which determines the expected structure of every available HID in the Surface Hub, and discovered that for touch input, every packet is expected to contain a maximum of 5 contact points. This is due to the Packet Reporting Mode of the Surface Hub.

Packet Reporting Mode

The Packet Reporting Mode[6] are the different ways in which touch data is sent to the operating system. The Surface Hub uses a hybrid mode. This means that every report contains less than the maximum number of contacts that the device supports. For the Surface Hub, every packet sent must contain a maximum of 5 contact points. As an example, if there are 15 active touches, the Surface Hub will send 3 separate packets to the operating system, each containing 5 touches.

Packet is not sent with the correct format

The reason why MTK is not able to detect all the touch input lies in how the Surface Hub sends its touch data to macOS. macOS expects to receive a packet that contains a maximum of 5 touches. However, our analysis showed that the Surface Hub sends out one packet containing all those touches in Guest Mode. As an example, if there are 15 active touches, a single packet that contains all 15 touches is sent to the operating system. This does not comply with the structure of the packet that macOS expects and the remaining touches gets ignored by macOS.

We tried to solve this issue

To solve this issue, we tried to modify Touch Up. Instead of using IOHIDManager, we tried using the IOUSBLib[7] li-

[6] https://learn.microsoft.com/en-us/windows-hardware/design/component-guidelines/touchscreen-packet-reporting-modes

[7] https://developer.apple.com/documentation/iokit/iousblib_h

brary. This library allows us to access hardware device interface, allowing applications to communicate and control the hardware from outside the kernel. We thought by using this framework, we would be able to read and interpret all the data that is sent through the interface ourselves, including the data that was ignored by macOS when using IOHIDManager. Unfortunately, this framework did not grant us access to the HID interface of the Surface Hub and we were not able to interpret the data coming in from the USB ourselves.

MTK can detect all touches on other touch displays

As mentioned earlier, we experienced this problem only occurred when working with the Surface Hub. From our analysis, MTK is able to take advantage of the maximum number of contact points on other touch displays that sends packets in the expected structure that is predefined by the touch display.

3.3.2 Improving Trace Visualization

Modify trace visualization

We modified how trace visualization works so that nodes only get processed when necessary. To do this, we implemented a queue which will contain nodes for trace visualization.

Queue for trace visualization

The queue contains nodes for trace visualization and is sorted in a timely order (see figure 3.4). Every trace contains this queue. The tail of the queue contains the node which shows the current position of the trace while the head of the queue shows the earliest recorded trace. Every node in the queue is attached to the scene and has a direct reference to its trace. This reduces the number of nodes that is being processed for trace visualization compared to the old implementation.

Delete nodes for traces that disappear

This is the current implementation of trace visualization. Before adding nodes to the scene, MTK checks if any traces disappeared from the last frame. When a trace disappears, all nodes in the queue of that trace gets deleted from the scene.

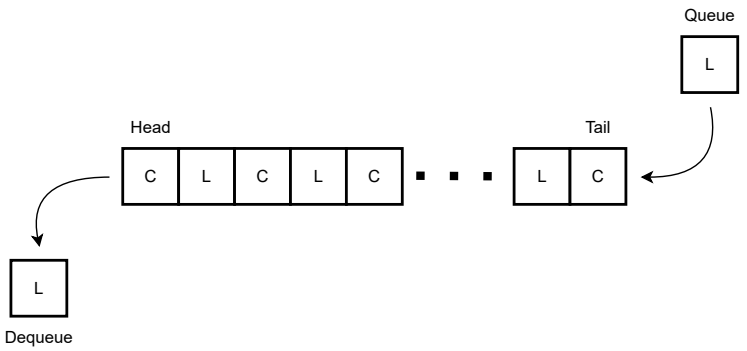

Figure 3.4: Queue for trace visualization. *C* represents a circle and *L* represents a line that connects together 2 circles, showing the path of the trace. An example is shown in figure 3.3

Add nodes for active traces

Next, MTK checks if there are any new traces or if a trace remains active. For new traces, a C node is pushed into the queue and added to the scene. For traces that remained active, an L node is pushed into the queue followed by a C node. Both nodes are then added to the scene. When the queue reaches its maximum capacity, the queue pops the first 2 elements and removes those nodes from the scene. Then, the latest L and C nodes are pushed into the queue and added to the scene.

Chapter 4

Evaluation

4.1 Measurement

Latency measurement

We performed a series of measurements to evaluate the end-to-end latency of the MTK. We referred to the method proposed by Casiez et al. [2017]. We decided to use this method due to the low cost and automation it provides. For this method, a vibration sensor is used to detect when to touch occurred and a photo-diode is used to detect when the screen responds. Latency is measured by calculating the difference between both occurrences. An image of the setup can be seen in figure 4.1.

Technical setup

The setup is slightly different compared to the proposed method. We used an Arduino Uno[1] instead of an Arduino Leonardo, which is also connected to a separate computer.

For the touch display, we worked with an 84" Microsoft Surface Hub from 2016. As we are using Guest Mode, the Surface Hub ran with a refresh rate of 60Hz. The touch display is connected to a Mac Mini which runs MTK. This version of the Mac Mini contains the M2 chip.

USB port can impact latency

We also considered the USB ports as a potential source of overhead. The Mac Mini has two USB-A and two Thun-

[1] https://docs.arduino.cc/hardware/uno-rev3

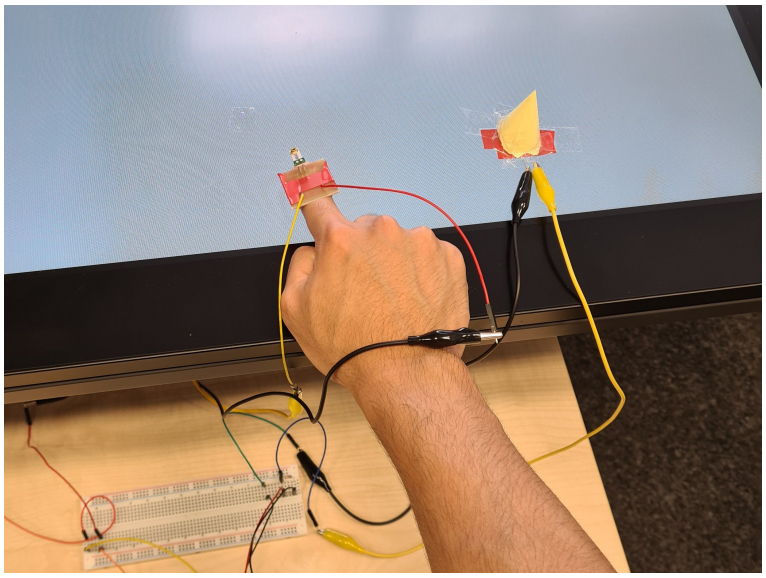

Figure 4.1: Technical setup to measure end-to-end latency. The vibration sensor is attached to the user's finger. The photo-diode is placed on the screen (right of the hand) and covered to prevent unwanted external light from interrupting the measurement.

derbolt 4 ports. However, the USB-A ports are connected together via a USB Hub which can cause a bit of latency [Ramadoss and Hung, 2008]. Although the effect is minimal, we connected the Mac Mini to the touch display via the Thunderbolt 4 port to avoid this overhead.

We measured latency for three kinds of interactions

For this evaluation, we measured the latency for three kinds of interactions: single touch, five touches, and tangible detection with 3 touches. Single touch will allow us to determine the base latency for a single touch. Five touches will allow us to determine the latency for the maximum number of touches that the MTK supports on the Surface Hub. This will also allow us to analyse the relationship between latency and the number of touches. To see the effects of the processing for tangible detection, we also measured the latency to detect PUCs, which have three contact points. We decided to measure the latency of detecting PUCs due to its simplicity.

For each of these interactions, we measured the latency with no trace visualization, with the old trace visualization and the new trace visualization. Latency with no trace visualization will represent the latency during normal use. We will also compare the latency of the old and new trace visualization to evaluate if there is an improvement. We did 100 touches for each trial, adding up to a total of around 900 measurements for the entire evaluation.

We measured latency for three types of visualization

For single touch, we measured the time difference between when a single touch occurs and when the screen reacts. For five touches, we measured the time difference between when the fifth touch occurs and when the screen reacts. The first four touches stayed active for at least two seconds for the buffer of all traces to reach its max capacity. For tangible detection, we measured the time difference between when the third touch occurs and when the screen reacts. The first two touches also stayed active for at least two seconds to fill up the buffer of the traces. The buffer size of trace can contain a maximum of 120 touches. All touches were performed with fingers.

Measurement approach

For latency measurement with single touch interaction, the buffer of the traces could not completely fill up with a single touch. However, we decided to conduct this measurement to form a baseline to compare with other interactions. Furthermore, this measurement can be used to analyse whether just enabling trace visualization could increase latency.

Buffer for single touch measurement is not filled

4.2 Results

The latency was calculated by subtracting the time when the vibration sensor detected a touch from the time the photo-diode reacted to the screen flash. Furthermore, 5.7 ms was added to due to the reaction time of the vibration sensor [Casiez et al., 2017]. The in-depth latency measurements for the all combinations of interaction and trace visualization can be seen in Appendix A.

With no trace visualization, we obtained an average latency

Latency with no trace visualization

of 94.97 ms for all three interactions. Latency for the three different interactions can be seen in figure 4.2. We observed that single touch interaction has the lowest latency out of the three. We also observe an increase in average latency of 2.16 ms with five touches compared to single touch. Furthermore, we observe an even slightly higher average latency of 2.58 ms with tangible detection compared to single.

Latency with old trace visualization

With the old trace visualization, we obtained an average latency of 107.56 ms for all three interactions. For all interactions, we observe an increase in average latency compared to without trace visualization (see figure 4.3). We observed a slight increase in latency with single touch (4.29 ms) and a noticeable increase in latency for both five touches (23.32 ms) and tangible detection (10.17 ms) compared to no trace visualization. Unlike without trace visualization, average latency with five touches is higher then tangible detection.

Latency with new trace visualization

With the new trace visualization, we obtained an average latency of 102.23 ms for all three interaction. Similar to the latency measured with the old trace visualization, we observe an increase in average latency compared to without trace visualization. We also observed a slight increase in latency with single touch (1.43 ms) and a noticeable increase in latency for both five touches (14.04 ms) and tangible detection (6.32 ms). However, this increase in latency is lower than the old trace visualization (see figure 4.3)

4.3 Discussion

Latency by receiving touch input via USB

The measurements showed the latency for three interactions with three ways of visualizing traces. Without trace visualization, we observed an average latency of 94.97 ms which is below 100 ms. This means that on average, the screen response will be perceived immediately in relation to when a touch occurs according to Bloch's law [Gorea, 2015] during normal use. However, with the new trace visualization, we observed an average latency of 102.23 ms which is slightly above the threshold of what Bloch's law considered something to be immediately perceivable. Both

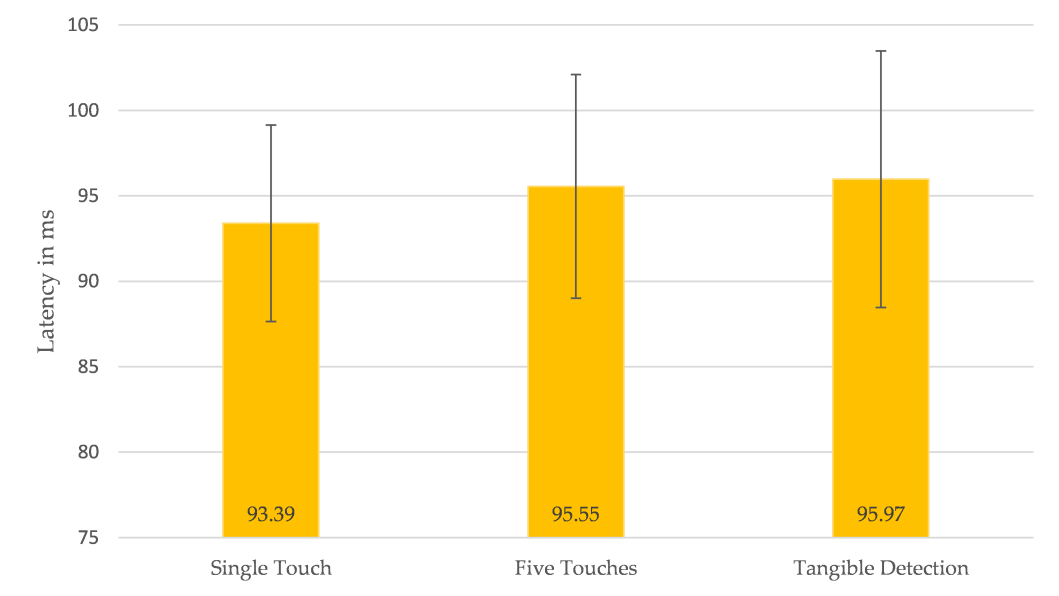

Figure 4.2: Latency without trace visualization for three interactions. Numbers represent the average increase in latency in milliseconds and error bars represent the standard deviation.

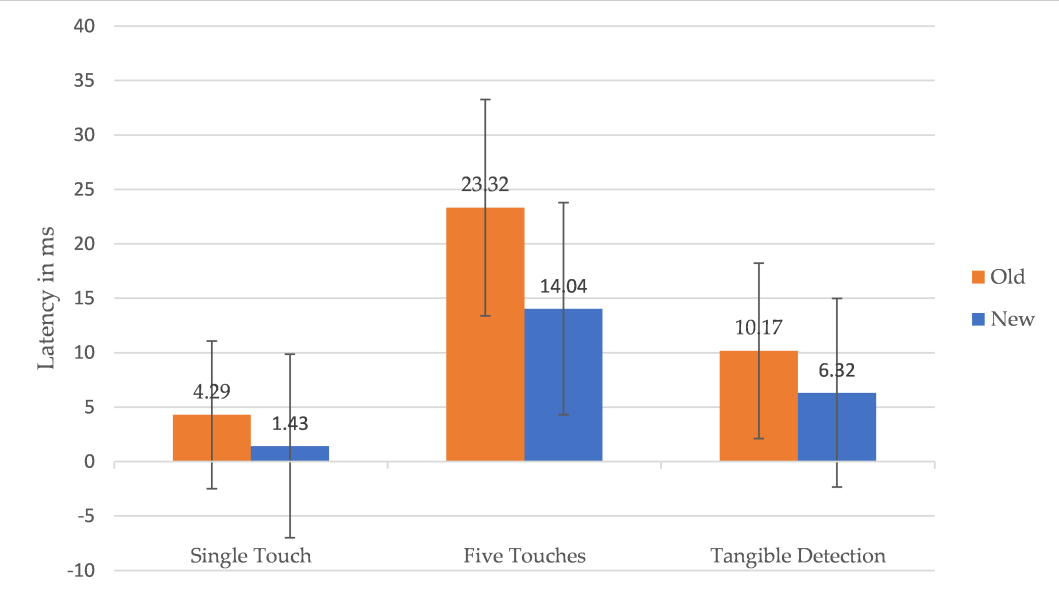

Figure 4.3: Increase in latency caused by old and new trace visualization for three interactions. Numbers represent the average increase in latency in milliseconds and error bars represent the standard deviation.

measurements are between 50 ms - 200 ms, which is the average latency of today's touch display [Ng et al., 2012].

Observation with no trace visualization

With no trace visualization, we observed a slight increase in latency with five touches compared to single touch. This was expected as more touch processing had to be done by the MTK, which increases latency. Furthermore, we also noticed a slightly higher average latency with tangible detection compared to five touches, although only three touches were being processed instead of five. This indicates that process of receiving touch input via USB caused less latency than the process for detecting tangibles. Overall, our implementation of receiving touch input via USB brought the latency to this level.

Latency comparison between old and new trace visualization

Comparing the the average latency of the old and new trace visualization for three interactions, we observed that the average latency of the new trace visualization is lower then the old trace visualization for all three interactions. This can be attributed to the reduction in the number of nodes that needs to be processed by the MTK compared to the old trace visualization. Overall, our aim to reduce latency in this part of the touch processing was a success.

Observation on latency with trace visualization enabled

Looking at the latency measurements with trace visualization enabled, we observed a slight increase in latency with single touch interaction, although no nodes for trace visualization were being processed before the touch occurred. This shows that latency increases by just enabling trace visualization. Unlike the latency measurements with no trace visualization, we observed a higher latency with five touches compared to tangible detection. This indicates that the trace visualization process is heavier than the processing of tangibles. Furthermore, we observed that for old trace visualization, the latency difference is 5.88 ms between single touch and tangible detection and 13.15 ms between five touches and tangible detection. We also see a similar pattern with the new trace visualization with the latency difference being 4.89 ms between single touch and tangible detection and 7.72 ms between five touches and tangible detection. This indicates that latency compounds the more touches there are and further supports the case that the process for trace visualization is heavy.

Range of latency is high

We also observed a fairly high range in the latency for every trial. An explanation for this situation could be caused by the rendering loop by SpriteKit. Due to the loop, it is possible for the touch to occur during a phase where touch isn't being detected by MTK. This means that additional latency was caused by waiting for the loop to go back to the initialization phase.

Type of tangibles used may impact latency

As mentioned earlier, MTK could only detect PUCs in this evaluation. Registering additional tangibles in MTK can possibly cause more latency. We did not perform latency measurements on PERCs as we focused mainly on improving the latency caused by the touch processing of the MTK, which PUCs suffices as a test object. However, it is possible that using PERCs could increase latency due to the overhead caused by the bluetooth communication between PERCs and MTK. Generally, we cannot conclude how much additional latency that the additional process for communication with PERCs causes.

Chapter 5

Summary and future work

5.1 Summary and contributions

We modified MTK

To summarize, we optimized latency in MTK based on the sources of latency that we found. To do this, we modified the way touch input is being received so that we can obtain touch input directly via USB. We also modified the way trace visualization works to improve latency when debugging.

Current latency of the MTK

We measured and evaluated the latency after modifying MTK. We observed an average latency of 94.97 ms with no trace visualization and 102.23 ms with the new trace visualization with all interactions combined. This level of latency can be attributed to receiving touch input via USB. Furthermore, the new trace visualization had less latency compared to the old trace visualization.

5.2 Future work

Find an alternate method for event distribution

There are some possible future works. Although we are getting an average latency of 94.97 ms with no trace visual-

ization, this amount of latency is relatively high compared to other devices [Casiez et al., 2017]. A possible reason is because of the way events are distributed in the MTK. As of now, an application needs its scenes to be registered in the MTKHub, which acts as an intermediary for sending events. Essentially, events are distributed to the scenes via the MTKHub. A possible future work is finding an alternate method such that the events can be distributed directly to the application instead of being delegated through the MTKHub. This could reduce the complexity of the framework, leading to a possible decrease in latency and setup process. However, we are not sure if this is possible without changing the architecture of the MTK.

Receive all available touch inputs via USB

As mentioned in section 3.3.1, we were not able to implement a method to detect all touches from the Surface Hub. A possible future work is to find a method to receive all the touch inputs from all touch displays via USB. This will allow users to take advantage of the maximum contact point that the touch display provides.

Reduce user's perception of latency

Another possible future work is to reduce the latency perceived in MTK. An example is predicting the direction of the touch during dragging tasks [Henze et al., 2016]. Overall, implementing this could make the system appear more responsive even though the actual performance remains unchanged.

Appendix A

Latency Measurements

Touches	Single Touch	Five Touches	Tangible Detection
Mean	93.39 ms	95.55 ms	95.97 ms
Standard Deviation	5.75 ms	6.54 ms	7.51 ms
Minimum	83.31 ms	80 ms	81.19 ms
Maximum	104.82 ms	107.73 ms	113.444 ms

Table A.1: Latency measurement with no trace visualization.

Touches	Single Touch	Five Touches	Tangible Detection
Mean	97.68 ms	118.87 ms	106.14 ms
Standard Deviation	6.78 ms	9.93 ms	8.06 ms
Minimum	85.44 ms	100.68 ms	82.87 ms
Maximum	120.72 ms	138.31 ms	124.98 ms

Table A.2: Latency measurement with the old trace visualization.

Touches	Single Touch	Five Touches	Tangible Detection
Mean	94.82 ms	109.59 ms	102.29 ms
Standard Deviation	8.43 ms	9.74 ms	8.66 ms
Minimum	80.07 ms	91.27 ms	80.53 ms
Maximum	113.67 ms	129.46 ms	118.27 ms

Table A.3: Latency measurement with the new trace visualization.

Bibliography

François Bérard and Renaud Blanch. Two touch system latency estimators: High accuracy and low overhead. In *Proceedings of the 2013 ACM International Conference on Interactive Tabletops and Surfaces*, ITS '13, page 241–250, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450322713. doi: 10.1145/2512349.2512796. URL https://doi.org/10.1145/2512349.2512796.

Géry Casiez, Thomas Pietrzak, Damien Marchal, Sébastien Poulmane, Matthieu Falce, and Nicolas Roussel. Characterizing latency in touch and button-equipped interactive systems. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, UIST '17, page 29–39, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349819. doi: 10.1145/3126594.3126606. URL https://doi.org/10.1145/3126594.3126606.

Christian Cherek, Anke Brocker, Simon Voelker, and Jan Borchers. Tangible awareness: How tangibles on tabletops influence awareness of each other's actions. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pages 298:1–298:7, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5620-6. doi: 10.1145/3173574.3173872. URL https://doi.org/10.1145/3173574.3173872.

James C. Craig and Gary B. Rollman. Somesthesis. *Annual Review of Psychology*, 50(1):305–331, 1999. doi: 10.1146/annurev.psych.50.1.305. URL https://doi.org/10.1146/annurev.psych.50.1.305. PMID: 10074681.

Clifford De Raffaele, Serengul Smith, and Orhan Gemikon-

akli. An active tangible user interface framework for teaching and learning artificial intelligence. In *23rd International Conference on Intelligent User Interfaces*, IUI '18, page 535–546, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450349451. doi: 10.1145/3172944.3172976. URL https://doi.org/10.1145/3172944.3172976.

Jonathan Deber, Ricardo Jota, Clifton Forlines, and Daniel Wigdor. How much faster is fast enough? user perception of latency & latency improvements in direct and indirect touch. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, page 1827–1836, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450331456. doi: 10.1145/2702123.2702300. URL https://doi.org/10.1145/2702123.2702300.

Tiffany Field. *Touch*. A Bradford Book. Bradford Books, Cambridge, MA, February 2003.

George W. Fitzmaurice and William Buxton. An empirical evaluation of graspable user interfaces: Towards specialized, space-multiplexed input. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, CHI '97, page 43–50, New York, NY, USA, 1997. Association for Computing Machinery. ISBN 0897918029. doi: 10.1145/258549.258578. URL https://doi.org/10.1145/258549.258578.

George W. Fitzmaurice, Hiroshi Ishii, and William A. S. Buxton. Bricks: Laying the foundations for graspable user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, page 442–449, USA, 1995. ACM Press/Addison-Wesley Publishing Co. ISBN 0201847051. doi: 10.1145/223904.223964. URL https://doi.org/10.1145/223904.223964.

Alberto Gallace and Charles Spence. The science of interpersonal touch: An overview. *Neuroscience & Biobehavioral Reviews*, 34(2):246–259, February 2010. ISSN 0149-7634. doi: 10.1016/j.neubiorev.2008.10.004. URL https://www.sciencedirect.com/science/article/pii/S0149763408001723.

Andrei Gorea. A refresher of the original bloch's law paper (bloch, july 1885). *i-Perception*, 6(4):2041669515593043, 2015. doi: 10.1177/2041669515593043. URL https://doi.org/10.1177/2041669515593043. PMID: 27433317.

Niels Henze, Markus Funk, and Alireza Sahami Shirazi. Software-reduced touchscreen latency. In *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services*, MobileHCI '16, page 434–441, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450344081. doi: 10.1145/2935334.2935381. URL https://doi.org/10.1145/2935334.2935381.

Hiroshi Ishii. Tangible bits: Beyond pixels. In *Proceedings of the 2nd International Conference on Tangible and Embedded Interaction*, TEI '08, page xv–xxv, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605580043. doi: 10.1145/1347390.1347392. URL https://doi.org/10.1145/1347390.1347392.

Bernard J. Jansen. The graphical user interface. *SIGCHI Bull.*, 30(2):22–26, apr 1998. ISSN 0736-6906. doi: 10.1145/279044.279051. URL https://doi.org/10.1145/279044.279051.

Blair A Jenkins and Ellen A Lumpkin. Developing a sense of touch. *Development*, 144(22):4078–4090, November 2017.

Topi Kaaresoja and Stephen Brewster. Feedback is... late: Measuring multimodal delays in mobile device touchscreen interaction. In *International Conference on Multimodal Interfaces and the Workshop on Machine Learning for Multimodal Interaction*, ICMI-MLMI '10, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781450304146. doi: 10.1145/1891903.1891907. URL https://doi.org/10.1145/1891903.1891907.

Scott R. Klemmer and James A. Landay. Tangible user interface input: tools and techniques. 2004. URL https://api.semanticscholar.org/CorpusID:108003762.

Rene Linden. Multitouchkit: A software framework for touch input and tangibles on tabletops and mobile devices. Master's thesis, RWTH Aachen University, Aachen, September 2015.

Shengmei Liu and Mark Claypool. The impact of latency on navigation in a first-person perspective game. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391573. doi: 10.1145/3491102.3517660. URL https://doi.org/10.1145/3491102.3517660.

I. Scott MacKenzie and Colin Ware. Lag as a determinant of human performance in interactive systems. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, CHI '93, page 488–493, New York, NY, USA, 1993. Association for Computing Machinery. ISBN 0897915755. doi: 10.1145/169059.169431. URL https://doi.org/10.1145/169059.169431.

Albert Ng, Julian Lepinski, Daniel Wigdor, Steven Sanders, and Paul Dietz. Designing for low-latency direct-touch input. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, UIST '12, page 453–464, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450315807. doi: 10.1145/2380116.2380174. URL https://doi.org/10.1145/2380116.2380174.

Albert Ng, Michelle Annett, Paul Dietz, Anoop Gupta, and Walter F. Bischof. In the blink of an eye: Investigating latency perception during stylus interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, page 1103–1112, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450324731. doi: 10.1145/2556288.2557037. URL https://doi.org/10.1145/2556288.2557037.

Lalitha Ramadoss and John Y. Hung. A study on universal serial bus latency in a real-time control system. In *2008 34th Annual Conference of IEEE Industrial Electronics*, pages 67–72, 2008. doi: 10.1109/IECON.2008.4757930.

Reiza Rayman, Serguei Primak, Rajni Patel, Merhdad Moallem, Roya Morady, Mahdi Tavakoli, Vanja Sub-

otic, Natalie Galbraith, Aimee van Wynsberghe, and Kris Croome. Effects of latency on telesurgery: An experimental study. In *Lecture Notes in Computer Science*, Lecture notes in computer science, pages 57–64. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

G. Robles-De-La-Torre. The importance of the sense of touch in virtual and real environments. *IEEE MultiMedia*, 13(3):24–30, 2006. doi: 10.1109/MMUL.2006.69.

Bertrand Schneider, Patrick Jermann, Guillaume Zufferey, and Pierre Dillenbourg. Benefits of a tangible interface for collaborative learning and interaction. *IEEE Transactions on Learning Technologies*, 4(3):222–232, 2011. doi: 10.1109/TLT.2010.36.

Andrew Sears. Improving touchscreen keyboards: design issues and a comparison with other devices. *Interacting with Computers*, 3(3):253–269, 12 1991. ISSN 0953-5438. doi: 10.1016/0953-5438(91)90016-U. URL `https://doi.org/10.1016/0953-5438(91)90016-U`.

Orit Shaer and Eva Hornecker. Tangible user interfaces: Past, present, and future directions. *Foundations and Trends in Human-Computer Interaction*, 3:1–137, 01 2009. doi: 10.1561/1100000026.

Kengo Somei, Toru Tsumugiwa, Ryuichi Yokogawa, Mitsuhiro Narusue, Hiroto Nishimura, Yusaku Takeda, and Toshihiro Hara. Effects of delayed visual feedback on brain activity: A near-infrared spectroscopy study. In *2023 IEEE/SICE International Symposium on System Integration (SII)*, pages 1–6, 2023. doi: 10.1109/SII55687.2023.10039114.

Dale Stack. *The Salience of Touch and Physical Contact During Infancy: Unraveling Some of the Mysteries of the Somesthetic Sense*, volume 2, pages 351 – 378. 11 2007. ISBN 9780470996348. doi: 10.1002/9780470996348.ch13.

Simon Voelker, Kosuke Nakajima, Christian Thoresen, Yuichi Itoh, Kjell Ivar Øvergård, and Jan Borchers. Pucs: Detecting transparent, passive untouched capacitive widgets on unmodified multi-touch displays. In *Proceedings of the 2013 ACM International Conference on Interactive Tabletops and Surfaces*, ITS '13, page 101–104,

New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450322713. doi: 10.1145/2512349.2512791. URL https://doi.org/10.1145/2512349.2512791.

Simon Voelker, Christian Cherek, Jan Thar, Thorsten Karrer, Christian Thoresen, Kjell Ivar Øvergård, and Jan Borchers. Percs: Persistently trackable tangibles on capacitive multi-touch displays. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, UIST '15, page 351–356, New York, NY, USA, 2015a. Association for Computing Machinery. ISBN 9781450337793. doi: 10.1145/2807442.2807466. URL https://doi.org/10.1145/2807442.2807466.

Simon Voelker, Kjell Ivar Øvergård, Chat Wacharamanotham, and Jan Borchers. Knobology revisited: A comparison of user performance between tangible and virtual rotary knobs. In *Proceedings of the 2015 International Conference on Interactive Tabletops & Surfaces*, ITS '15, page 35–38, New York, NY, USA, 2015b. Association for Computing Machinery. ISBN 9781450338998. doi: 10.1145/2817721.2817725. URL https://doi.org/10.1145/2817721.2817725.

Malte Weiss, Roger Jennings, Ramsin Khoshabeh, Jan Borchers, Julie Wagner, Yvonne Jansen, and James D. Hollan. Slap widgets: Bridging the gap between virtual and physical controls on tabletops. In *CHI '09 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '09, page 3229–3234, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605582474. doi: 10.1145/1520340.1520462. URL https://doi.org/10.1145/1520340.1520462.

Index

Typeset December 6, 2023