Spring 2018

An Implementation and Usability Study of a Natural User Interface Virtual Piano

Zackery Frazier
zmf9@zips.uakron.edu

Table of Contents

Abstract

We present an implementation of a Natural User Interface (NUI) virtual piano keyboard. Using the Unity game engine and a Leap Motion hand controller, users are able to use their hands to interact with and play music in a virtual scene. Various approaches were attempted in refining the user experience of the virtual piano, and the successes and shortcomings of each implementation method are outlined and elaborated upon. Additionally, the most successful method, which was achieved by using Unity's physics engine to detect rigid body collisions between virtual fingers and keys, was used to complete a usability study involving both experienced and amateur pianists. Subjects were presented with a series of playability tasks designed to gauge their ability to accurately use the virtual keyboard. Both the usability test data and the subject's personal feedback were collected and analyzed.

Project Scope

In approaching my Honors Project, I was heavily influenced by my project sponsor, Dr. Xiao. His affinity with modern Human Computer Interaction methodologies in several of his classes exposed me to the Leap Motion sensor, which spurred my interest in learning how to develop applications with this new device. For a term project in his Human Computer Interaction class that I took in Spring 2017, I implemented a simplistic virtual piano that users could interact with using the Leap Motion controller. This primitive implementation relied on Unity's physics engine to detect collisions between rigid hands and keys. However, in completing the project I was not satisfied with the original keyboard's responsiveness and performance. On weaker computers it was hardly usable to reliably play single keys at a time, it missed many user attempts to interact with the keyboard, and would sometimes outright crash if a user moved their hands too rapidly through the keys. Even on more powerful computers, interacting with several keys at the same time to play a piano chord, or even attempting a Glissando (The piano technique of running your hand across the keyboard to play every key in a row) were entirely impossible. I expressed these concerns to Dr. Xiao and he suggested that I experiment with different implementations of a virtual piano and carry out a performance comparison on random users in order to ascertain the usability of each approach. This proposition intrigued me, and I set out to refine my original implementation while exploring alternatives. This report will focus on the optimizations made on the original keyboard scene, an outline of the successes and failures of the other explored methods of implementation, and the resulting data and concluding analysis of the usability study while comparing the original and optimized keyboard scenes.

Human Computer Interaction

Human Computer Interaction (HCI) is an area of study focused on the the interaction between humans and computers through a user interface. There are several mediums through which humans can interact with the computers. These include a Command-Line Interface (CLI), a Graphical User Interface (GUI), and a Natural User Interface (NUI). A Command Line Interface is a text-based command prompt with which users must manually enter commands via keywords. Graphical User Interfaces offer users a way to execute commands via interaction with graphical images and visual icons. Natural User Interfaces, a recent trend in the area of HCI, allows users to user their voice or hand gestures as input for issuing commands. Modern NUI methods require specialized devices for capturing and understanding this variety of input.

Hand Gesture Recognition

Hand gestures can be used to produce interactions between humans and computers. Hand gesture recognition requires a capture device, such as a RGB camera, an infrared camera, or instrumented gloves. These items capture data regarding the image, position, and rotation from the user's hands. Hand gesture recognition is typically classified into two categories; static gestures and dynamic gestures. Static gestures are used to describe hand poses while dynamic gestures are used to identify hand movements[2].

Overview of Unity3D

This project focuses on using Unity3D, a game development engine to develop the virtual piano. Unity3D is a cross platform development environment, supporting Windows, Mac, Linux, and other mobile platforms such as iOS and Android. The extensive development community endows Unity developers with a plethora of open source models, scripts, and music to assist in development. The highlight of Unity development environment relates to the provided physics engine, which allows developers to easily simulate real world physical interactions between scene components. Unity ships with MonoDevelop, a cross-platform IDE tailored to C# development. Users can write custom C# scripts to control the scene assets and direct the flow of the application.

Overview of Leap Motion Controller

The Leap Motion controller is a sensor device that tracks human hands. It was developed by Leap Motion after being announced as a concept in May 2012. The company set out to create a portable, user-friendly, and cost effective device that would provide users with touchless control of computer software. The device integrates with user's computers via USB 2.0 port and contains two optical sensors and three infrared LED diodes[1]. These components perceive a field of view of 150 degrees, and works optimally within the ranges of 25 to 500 millimeters above it[1]. The raw input of the Leap Motion controller is a set of visual silhouettes represented as varying IR brightness metrics[1]. The device contains proprietary software that combines the sensor data with a standard model of a human hand to maximize tracking accuracy. Various corrections of image distortion are applied via shaders and bilinear interpolation[1]. The optimal distance for hand detection can be seen in Figure 1, while the bounding interaction box surrounding the device is displayed in Figure 2.

Figure 1. Optimal Hand Distance

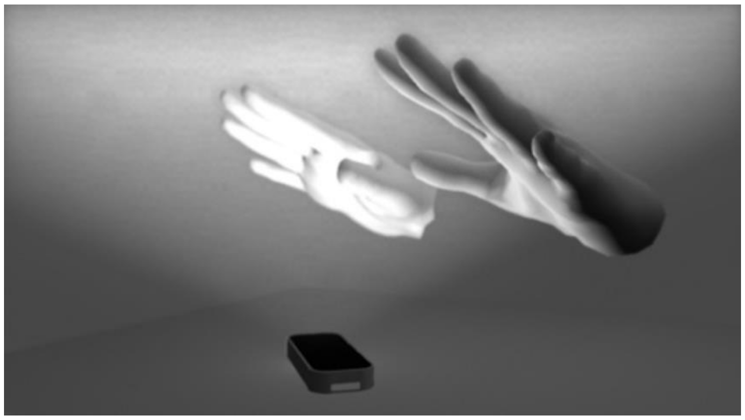
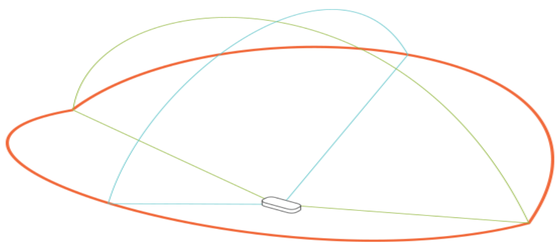


Figure 2. Leap Motion Interaction Box Visualization

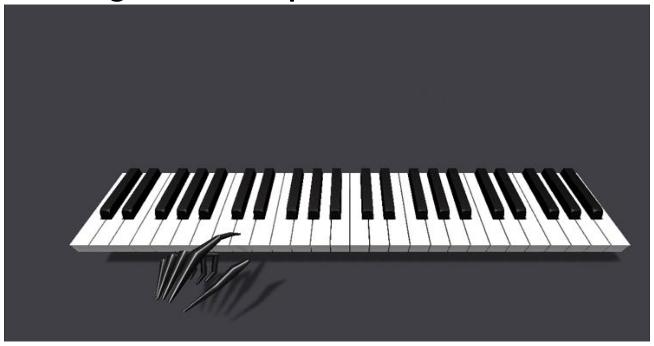


Implementation Details:

HandController (GameObject)

The HandController GameObject was key in the implementation of the virtual piano. Provided as a prefab in the Leap Motion SDK, it functions as the focal point for rendering virtual hands in the Unity scene. Developers can set properties appropriately to suit their individual needs, including which hand models to render, whether or not the hands are rigid, among others. While active, the HandController script includes the 3D hand model's positions, orientations, and textures in the scene based on the presence of hand data streamed from the device's area of perception. It is visualized as a clear box in the scene view, allowing the developer to specify a range within which hands can be rendered. A sample hand model is shown in Figure 3.

Figure 3. Example Hand Model



Keyboard (GameObject)

In order to build the virtual keyboard, CAD 3D design software was used to build models of both black and white keys. Once designed, they were exported to .fbx file format and imported into Unity. Four octaves worth of keys were included in order to mimic the presentation of a real life keyboard as closely as possible. The movement and collision of the keys is directed by KeyCollisionManager.cs script.

Light (GameObject)

The scene includes three Light objects, positioned triangularly around the keyboard. This was done to provide the user with clarity when viewing the keyboard, while also providing realistic shadows surrounding the keys to give the scene a more realistic feel.

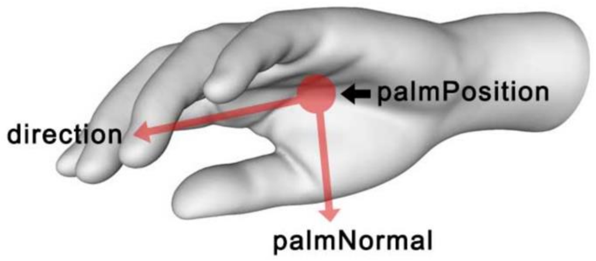
Camera (GameObject)

The scene utilizes a single Camera, positioned vertically above the piano key models, between the HandController area of perception. This provides an optimal viewing angle to allow users to perceive depth and understand the position of their virtual hands in relation to the keyboard.

HandController (Script)

Also included in the Leap SDK, this script is used to direct and access information regarding the detected hands during the duration of the scene. It contains useful methods for attaining the location, speed, and orientation of the perceived hand models, as well as determining the normal of the user's palms as seen in Figure 4. Hand models are composed of a hierarchy of sub-models that compose the overall Hand object. These include finger models, which exist to ease the access of developers to the individual fingers of the input hand data. Furthermore, these Finger objects are each composed of four Bone objects, which can again be accessed to the needs of the developer.

Figure 4. Palm Normal On Hand Model



KeyCollisionManager (Script)

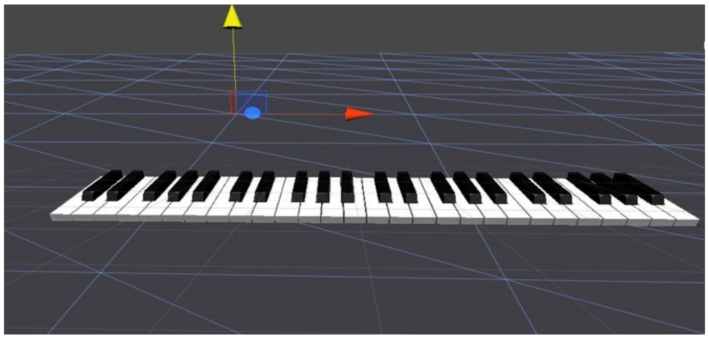
This script encapsulates most of the optimizations made to enhance the usability of the original application. It contains methods which carry out the algorithm described Figure 5:

Figure 5. KeyCollisionManager Pseudo Code

```
while hands are in frame:

    rigid body collision detected:

        if key-to-key collision:
            ignore

        if hand-to-key collision:
            play desired key
            interpolate key movement and play audio
            pause global rigidity of finger for 50ms OR
              until palm normal crosses 30 degrees
            ignore adjacent key collisions for 65ms
            return key to original position

        if collided finger goes below keys:
            block rigid finger with rigid plane
```

The event loop begins when the hand controller detects a user's hands. Once the loop has begun, the application listens for any and all rigid body collisions in the scene. Once a collision is detected, it is ignored if it is between two keys. Once it has been confirmed that the detected collision is between a key and a finger, a host of functions are executed. The audio for the particular key is played and the movement and color change of the key is interpolated. Next, the rigidity of the finger which was used is paused for either 50 milliseconds or until the palm normal exceeds 30 degrees in the vertical plane. This is to ensure that stray movement of a finger isn't mistakenly perceived by the application as an attempt to play another key, as user's tend to drag their fingers up through adjacent keys after playing a single key. Additionally, to curb the accidental collisions when the user is returning the finger to a playing position, the application ignores collisions between the used finger and keys adjacent to the selected key for 65 milliseconds, regardless of palm normal. Finally, a rigid plane is placed halfway between the top and bottom of each key, which prevents the user’s virtual hands from going below the keys (Figure 6).

Figure 6. Rigid Plane Halfway Through Keys



Explored Implementation Methods

Minimum Finger Velocity Threshold:

The initial idea with this approach was to ignore any key-to-finger collisions that occur if the finger is not moving at some minimum velocity moments before collision with a key. The HandController class allows the application to fetch the current movement speed of a given hand, finger, or bone in the scene. This approach attempts to avoid stray finger-to-key interactions that occur the user is attempting to play a single key. The problem with this approach lies in the fact that when attempting to play both a real or virtual piano, it was found that users don’t typically always make quick enough of a finger movement to generate a detectable increase in finger velocity. Rather, depending on the key transitions, users often gently glided their fingers to the desired key, meaning these interactions went ignored. More importantly, and something that will be touched upon when discussing the qualitative feedback received from the test subjects, is their report that subjects had an easier time playing the virtual keyboard when focusing on translating their entire arm to play a key, rather than just isolating the movement around their fingers and wrist. Additionally, constant requests for the current velocity of any given finger tip proved to be extremely taxing on the performance of the scene, and ultimately led me away from continuing with this approach.

Ray Casting:

Unity supports a method known as *RayCast*, which allows developers to specify a point of origin, a direction, and a maximum travel length for an unseen vector to be virtually “shot” in the scene, providing contextual information regarding any entities the ray collides with. The original idea was to extend RayCasts from the fingertips of a user such that downward movement of a finger would trigger a ray to be cast, return which key it collided with, and play the key appropriately. Ultimately, this approach ran into a similar issue mentioned above in the Minimum Finger Velocity Threshold section. Users did not often move their finger with a quick and direct enough initial movement to be able to reliably tell when they were trying to play a single key with any given finger. This approach also had problems accounting for cases where the user’s finger was not moving straight downwards when positioned vertically above the desired key. If a user were to transition between two keys that had several keys between them, most of the detected finger motion was made in a horizontal direction, while the faint amount of vertical translation needed to be detected to press the key often went unperceived. Finally, the biggest problem with this method is that it made it difficult for users to sustain a note. The heuristics needed to determine whether or not a user was attempting to hold a note for an extended amount of time when applying a RayCast-based approach ultimately dissuaded me from using RayCasting to implement a virtual piano.

Usability Testing Methods

After getting to a satisfactory point with the responsiveness of the optimized piano scene, the next task was to run usability tests in order to gather quantitative and qualitative data regarding how users performed using the virtual piano. Voluntary test subjects were given a set of tasks to complete and then were asked to give personal feedback regarding the feel of the application. The trials are as follows:

One Octave, Varying Increments:

Users were tasked with playing eight keys within the same octave in ascending order at different increments. The first trial was one key at a time, the second was two at a time, the third was three at a time, and the fourth was four at a time. Users were able to use whichever fingers they felt most comfortable with for each key.

Quick Selection:

Users were flashed a random key within the range of one octave. Their goal was to then hone in on that key and play it as quickly as possible with minimum time for gauging their hand's depth within the scene. Ten trials were performed, and trials were judged on their success or failure depending upon the accuracy of their selection. Users were not restricted regarding which finger they could use for each key.

Quick Selection Chords

Similar to the quick selection test mentioned above, however users were instead flashed a specific piano chord (A triad of three different notes within an octave). Five trials were done.

One Octave, Varying Increment Test Data

Pianist Subject	8 Keys, 1 at a Time	8 Keys, 2 at a Time	8 Keys, 3 at a Time	8 Keys, 4 at a Time
A	Success	Success	Success	Failure
B	Success	Success	Success	Failure
C	Success	Success	Failure	Success
D	Success	Success	Failure	Failure
E	Success	Success	Success	Success

Amateur Subject	8 Keys, 1 at a Time	8 Keys, 2 at a Time	8 Keys, 3 at a Time	8 Keys, 4 at a Time
A	Success	Success	Failure	Failure
B	Success	Success	Failure	Failure
C	Success	Failure	Failure	Failure
D	Failure	Success	Success	Failure
E	Success	Failure	Failure	Failure

Quick Selection, Single Keys Test Data

Pianist Subject	Quick Selection Trial (Single Key)
A	8/10
B	10/10
C	10/10
D	9/10
E	10/10

Amateur Subject	Quick Selection Trial (Single Key)
A	7/10
B	9/10
C	7/10
D	10/10
E	8/10

Quick Selection, Piano Chord Test Data

Pianist Subject	Quick Selection Trial (Piano Chord)
A	3/5
B	4/5
C	3/5
D	2/5
E	2/5

Amateur Subject	Quick Selection Trial (Piano Chord)
A	0/5
B	1/5
C	0/5
D	0/5
E	0/5

Usability Test Data Insights

As a general trend, it's worth noting that throughout every test, experienced pianists had more success using the virtual keyboard than the amateurs. This is less apparent in the first two tests. However, the amateur pianists struggled with successfully playing piano chords, as only one subject was able to do so.

One Octave, Varying Increments:

When asked to play an octave's worth of keys in a row using increments of two, the pianist group had all subjects able to pass, while only two in the amateur group were unable to complete this task. Amateur success tapers off drastically during the three and four incremented trials, with none of the subjects able to successfully complete the four-fingered task. Conversely, while experienced pianist success falls off during the three and four incremented trials, 3/5 and 2/5 successes, respectfully, it's worth noting this as an area of future improvement for the project. I'd like to explore different ways for the application to handle multi-finger interactions with the keyboards to enhance the general usability.

Quick Selection:

In comparing the pianists and amateur results, the single key quick selection data reassures my confidence in the responsiveness of my application regarding playing individual notes one at a time. With such a close average number of successes, 8.2 for the amateur group and 9.2 for the pianists, most users reported this to be the easiest and most intuitive task.

Quick Selection Chords

This test highlights the largest disparity between the experienced and amateur pianist groups. With only a single amateur pianist able to play a piano chord on one occasion, and experienced pianists only able to play a piano chord on an average of 2.8 times, this is clearly an area of improvement that should have highest priority.

Subjective User Feedback

After completing the usability testing, subjects were asked to give any personal feedback and input regarding their experience using the virtual piano. This was done to gauge the general "feel" of the application, and will be used to further improve the responsiveness and guide future work.

The most common complaint of the experienced pianist group relates to the actual wrist and finger movement required to play a key. When playing a real piano, inexperienced musicians are encouraged to focus most of the movement at the wrist and finger tips, while minimizing the total translation of the entire forearm. However, users reported that isolating their movement to the wrist and fingers often caused them to accidentally play incorrect keys, or miss a key entirely. Rather, they reported that they had the easiest time completing the tests while moving their entire arm in the vertical plane after they had positioned their wrists and fingers directly above the desired keys. Because this was communicated nearly universally by all of the experienced pianist subjects, this feedback is an intriguing insight that highlights the existing differences between playing the virtual piano compared to a real piano.

Comparatively, both groups of pianists commonly reported that the biggest problem they had in accurately using the virtual keyboard was the application perceiving them playing a key directly adjacent to their target key. Upon logging the positional data of the user's hands and requesting they try to recreate this problem, it was found that users would often begin to play a key with their fingers in the correct vertical position, but upon moving their finger downward to interact with the target key, their finger tended to slightly shift horizontally just in time for the rigidity of their finger to be restored, causing the application to perceive them trying to play the adjacent key. Adjusting the length of time in which the application ignores collisions with keys adjacent to the target key could help smooth out this issue.

Future Work

Going forward, enhancing the user's ability to play piano chords is of the highest priority, as this change would deliver a large amount of value to the application. Additionally, somehow reducing the number of incidental collisions with keys adjacent to the target key would assist in the usability of the application. Allowing users to record and store video of the recent interactions would be a great quality of life addition as well.

References

[1] "API Overview" *Leap Motion Developer*, developer.leapmotion.com/documentation/java/devguide/Leap_Overview.html. April 2018

[2] Reifinger, Stefan, et al. "Static and Dynamic Hand-Gesture Recognition for Augmented Reality Applications." *Human-Computer Interaction. HCI Intelligent Multimodal Interaction Environments Lecture Notes in Computer Science*, pp. 728–737. April 2018