VRTyper: Keyboard for Virtual Reality Add On

Kylee Krzanich
Stanford University
450 Serra Mall, Stanford, CA
krzanich@stanford.edu

1. Introduction

Virtual Reality is often touted as the future of schools, businesses, and social gatherings. People will soon be able to use the platform to learn in a virtual classroom or conduct meetings in a virtual conference room. However, the ability to interact with the virtual environment is paramount to Virtual Reality being widely adopted across these areas. Users need to be able to do things like write and read with the same ease with which they carry out these basic tasks in the physical world. One large barrier to converting these physical spaces into virtual ones is simply one's ability to type. In a virtual conference room, you would most likely have your computer in front of you in order to take notes. How can you take notes while you're wearing a headset? Do you type on a real keyboard that is projected into the Virtual Space so that you can visualize it? Do you type freely on a keyboard that's projected anywhere onto the scene? How do we simulate a real keyboard without additional bulky hardware? Answering these questions will help to facilitate the adoption of Virtual Reality within more disciplines than just gaming, and increase the variety of situations in which it's beneficial for people to interact with each other virtually. We sought to create a simple, low-cost keyboard without the lag usually associated with VR tracking.

1.1. Summary

We created a low-cost virtual keyboard for easy typing within the VR space. The goal of our project was to make virtual typing a comfortable, accurate, and realistic experience. We also aimed to minimize the cost and bulkiness of any additional hardware. We implemented a proof-of-concept keyboard and demo in Unity that can be easily integrated into any system. In most VR setups, the user has many controls and props that they must hold. Additional gloves and tracking equipment would further hinder the user's free movement and doesn't align with our goal of having fluid integration of the keyboard and typing experience. Current solutions to this problem offer relatively complicated systems including gloves with haptic feedback. As opposed to the relatively complicated systems currently being offered as solutions, like gloves with haptic feedback, our solution allows the user to start typing without having to blindly put on gloves while wearing a headset.

Instead, our keyboard relies solely on the use of Leap Motion for hand recognition and fingertip location tracking. Whereas using the HTC Vive relies on multiple photodiodes on an object to get an accuracy of 2 mm due to the jitter from the lighthouse and a latency of 22 ms [5], the Leap Motion controller allows us to get extremely precise fingertip location of within 1 mm accuracy and sub 6 ms latency [11]. The Leap Motion controller also has a built in capability for recognizing hands and fingers extremely accurately within a 1 mm range. For now we project the keyboard in the air approximately 300 mm away from the user, however, our plans for future prototypes will use the Leap Motion camera to detect and measure tables so that we can project the keyboard onto a flat surface, which we believe will create a more natural typing experience for users.

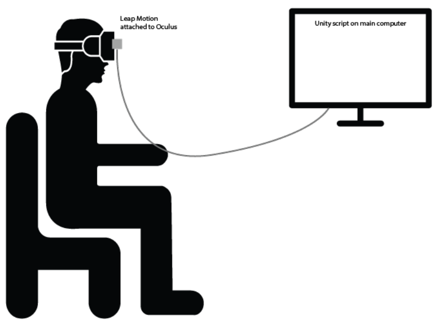


Figure 1: This is a sketch of the full system. The Leap Motion is mounted to the HMD and then connected to the computer to transfer positional data.

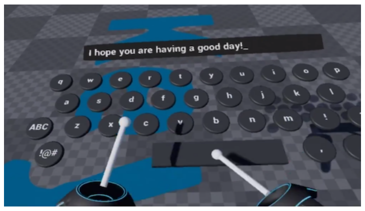
We evaluate our keyboard based on three criteria: (1) a simple yet realistic user experience, which will be assessed with user studies; (2) accuracy of finger location and latency of the entire system; and (3) the total cost and additional hardware bulk of the system.

1.2. Context

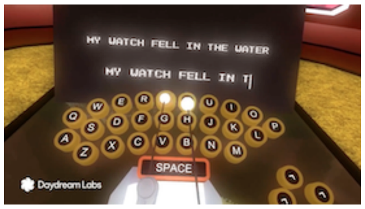
Our final product is the realization of a universally compatible Virtual Reality keyboard. We can add the additional hardware to any existing system for a fixed, low cost. The keyboard itself can be integrated into games or administrative controls. When users are setting up their VR system, they typically have to enter in their name using a point and click remote. With our keyboard, we hope that users being able to type out their information in a familiar manner will legitimize Virtual Reality as a real solution rather than simply a gimmick.

2. Related Works

Most of the existing and well-known VR keyboards are point and click style as seen in figure 2. The HTC tracking system relies on using Infrared light to track photodiodes on each of the remotes [2]. This means that the Vive controllers can very accurately point and click on each letter but the set up very much resembles a gaming platform and is not convenient for typing anything longer than a single sentence [1].



(a) HTC style keyboard



(b) Drumstick style keyboard

Figure 2: Above the reader can compare two keyboards with controller interfaces: (a) The first image depicts a $14.99 keyboard available for typical HTC setups. Users use the remote to point and click on small letters from a distance. (b) The second image depicts Google's "drum set" keyboard where users instead treat the controllers as long sticks to bang on the keys.

Google recently released their own version of the HTC keyboard that allows users to use to beat on the keyboard with drumsticks [7]. They claimed that this was much easier for people who had trouble holding their controller steady in the previously discussed method.

There is a considerable amount of existing work toward creating gloves that can mimic an actual keyboard. For example, Natoli *et al.* [8] describes how to implement a glove-based mapping using a neural network for recognizing the glove motion. Similarly, the work by Wu *et al.* [12] touches on the importance of the sensory experience within VR. Their paper describes a haptic feedback glove with micro-speakers that act as actuators to give the impression of a keystroke. Through multiple user studies, they showed that even without haptic feedback, users preferred using purely virtual keyboards over virtual keyboards overlaid on top of physical keyboards [12], which introduces unnecessary equipment into the scenario. The result of their research demonstrates the need for a simple, accurate keyboard. In extending the principle of simplicity explored in their findings, our implementation won't require the user to put on gloves or any other equipment because we believe it detracts from the experience and any physical component the user is required to interact with would be subject to wear and tear, which may significantly increase the expected cost.

We take inspiration for future prototypes from, among others, the work of Lin *et al.* [6], which describes the keyboard visualization process. Their research details the method they used to determine the spatial layout of the world around the user. We hope to apply similar techniques to project our keyboard onto surfaces in the space.

The work by Du *et al.* [3] is the most similar to our intended project. However the keyboard itself isn't projected in virtual reality, it's simply projected onto the surface of the table and is meant to overcome bulky keyboards with lots of wiring. Their keyboard also uses the Swissranger SR-2 which has an accuracy anywhere from two centimeters down to a few millimeters, depending on the scenario [4]. According to their analysis, the keyboard had a false detection rate of up to 8.5% [3]. We aim to produce a keyboard with a significantly lower false detection rate, which we think is achievable based on the precision of the hardware we are using to detect finger position and motion.

3. Methods

The project consists of four overarching parts: creating the Unity application and graphics objects, setting up and configuring the Leap Motion for our project specifications, defining the behavior for the keyboard interaction, and optimizing the application through directed user studies.

3.1. Creating the Unity Application

First, we needed to create the Unity application and corresponding graphics and interactions objects for the demo. For reference, we used Unity 2018 Version 3.7f along with the Leap Motion Core Assets Version 4.4 and Leap Motion Interaction Engine version 1.2.

The applications consists of the following integral components: the stage, the keyboard, the hand models, the Leap interaction manager, and the Google cardboard camera and stereo render.

Through studying the documentation and examples included with the Leap Motion Interaction Engine package, we set up our stage which included the platform, lighting, and, most critically, the Text Mesh object to be modified by the keyboard. Also with reference to the Leap demos [10], we added a Leap Rig, which serves are the VR scene; the main camera, hand models, and hand interaction manager are all contained within the Leap Rig. The hand models set the graphics and movement for the tracked hands, for which we use a Low poly model to ensure quick rendering and thereby reducing latency, and the interaction manager allows us to set the active finger tracking and contact settings.

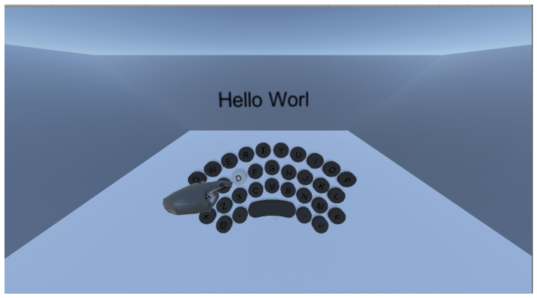


Figure 3: This image shows the entire scene that we created in Unity. The barriers on the space were meant to give the user depth cues and the only visual objects in the scene were the hands, keyboard, and text.

For the keyboard buttons themselves, we found an open source keyboard on Github that was designed with HTC Vive controllers in mind [9]. We manually overlaid a canvas to display the keyboard characters and then transferred the models into UIbutton objects so that we could construct interaction callbacks. See figure 3 for the full scene layout.

Though we initially began the project with a single main camera, for the final step, we needed to incorporate the Google Cardboard SDK to give users better depth perception through the Google Cardboard Camera. There was some difficulty getting the Google Cardboard Camera integrated in the existing project. Ultimately, integration process involved disconnecting the Leap Rig camera, and attaching the Google CardboardMain object to the Leap Rig, and then setting the Google Cardboard Camera SDK as the default XR Leap Manager.

3.2. Configuring the Leap Motion

Next, we needed to configure the Leap Motion to track our hand models and facilitate the interaction with the keyboard. Setting up the Leap Motion consisted of downloading the newest SDK and using the calibration script in order to ensure as much fingertip accuracy as possible.

After setting the ideal interaction distance for the Leap Motion, initially through trial and error and some basic measurements, the Leap Motion seamlessly integrated with our Unity project containing Leap Motion assets on Windows.

3.3. Keyboard Interaction

In order to define our own button protocol, we had to make each key its own UIbutton which we could then link to our Interaction Behavior script to detect the letter and update the text on the screen accordingly.

Our Interaction Behavior script consisted mainly of scheduling lambda functions within the onPress button event for each of the keys, to appropriately modify the text (or in the case of the shift key, modify the behavior of the other keys). As a result of user study 1, which indicated the benefit of sound effects, we added button press sound effects to the onPress event functions.

Further, as another result of user study 1, we added hover indicators to each of the keys by attaching the Simple Interaction Glow from the Leap Motion Interaction Engine package to each of the button objects.

Finally as a result of user study 3, we configured the position and rotation of the keyboard to facilitate a smoother user interaction.

3.4. User Studies

As alluded to in previous sections, a critical component of our project was the design and execution of three user studies. The first of which was designed to collect early feedback from users on the environment and interaction during development in order to expedite the modification process. The second user study focused on obtaining benchmarks of the VRTyper along standard VR keyboard metrics, namely, error rate, false detection rate, and typing speed. The third user study was designed in an attempt to further improve and refine the typing experience by testing six variations of the position and rotation of the keyboard in the VR space.

Each study was conducted with four participants. Study 1 gave the participants no time to practice before the study; study 2 gave the participants 10 minutes each to practice with the keyboard prior to the study; and user study 3 was conducted with the same participants as study 2, so no practice time was given. Additionally, the six different configurations of the keyboard presented in study 3 were presented to the different participants in independently, randomly selected orders so as to reduce the effect of novelty or fatigue.

The outcomes of all three of these studies are detailed in the results section.

3.5. Challenges

The first difficulty came in the form of software compatibility. The libraries and SDKs of both Unity and Leap Motion are somewhat notoriously under-documented, and are updated frequently, causing backwards compatibility issues. Additionally, the Leap Motion Orion SDK is not currently compatible with MacOS, which made testing difficult.

We did obtain a Windows computer during development, which allowed us to test and develop using the Leap Motion, but ran into a second issue when we were unable to use the LCD screen in the View Master VR Headset with the Windows computer, as we found ourselves unable to adjust the frame rate from that computer. After much troubleshooting, we were finally able to adjust the frame rate by determining whether the GPU or CPU controlled the monitor (the CPU in our case), and then adjusting it from the appropriate control panel.

Another challenge we faced was in determining the optimal hit box for the button interactions, in order to maximize the likelihood of detection but minimize false positives or errant keystrokes. We solved this issue through a trial and error approach.

Finally, we encountered some difficulty in integrating the Google Cardboard SDK into our existing Unity project, mainly due to lackluster documentation for the Google Cardboard SDK.

4. Results

Our results are comprised of findings from the three user studies we ran, collecting user feedback on the environment and interaction, benchmarking standard VR keyboard metrics, and optimizing position and rotation configurations of the keyboard.

4.1. User Study 1

The first user study we ran was to collect immediate user feedback on the environment and interaction for use during the continued development process. This study was vital to our initial development and served as the basis for many of the features present in the final product.

We had participants type 20 words on a standard keyboard and then had them write 20 words with the VRTyper prototype. Users had three main complaints:

1. It was near impossible to tell whether your hand was correctly hovering over the letter you were typing.

2. It wasn't immediately clear (i.e., before looking at the changing text) whether or not they actually hit the key because there was not enough feedback.

3. The vertical keyboard didn't feel like a natural scenario.

Using the results from this user study we made several improvements: we rotated the keyboard 12 degrees so that the top keys were farther from the camera view than the bottom keys—we hoped this would provide a slightly more intuitive and natural typing environment; second, we added a highlight script so that users would know whether or not they were hovering over the intended key; and lastly, we added an audio sound effect to establish a more feedback rich environment for users.

4.2. User Study 2

Next, we ran a second user study, this time focusing on error rate, false detection rate, and typing speed, which are standard metrics for evaluating VR keyboards. We had a total of four participants who were able to use the VRTyper for up to 10 minutes for practice before we observed them typing 20 predetermined letters. See the results in figure 4.

User Study 2			
Participant	Error Rate	False Det Rate	Speed
User 1	0.3	0.2	3:36 min
User 2	0.2	0.15	3:15 min
User 3	0.3	0.15	3:41 min
User 4	0.25	0.35	2:59 min

Figure 4: This was a general user study to evaluate common benchmarks for our keyboard. As seen in the table, there were four users that were measured based on the error rate, false detection rate, and speed.

We asked users to focus more on accuracy than on speed, so the recorded times for each user are far below what is considered an ideal typing speed. Many users voiced their concern that the depth was difficult to discern, so they were not sure if they were too far from they keys or even if they had gone through the keyboard entirely. For this reason, we further modified our demo by adding in the Google Cardboard SDK, so that users could have more and better depth perception.

4.3. User Study 3

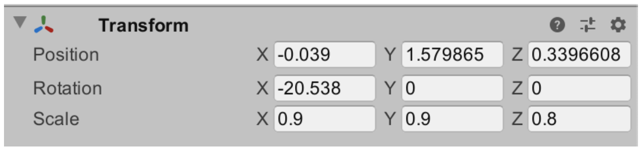
Finally, we ran a third user study testing specific positions and rotation angles of the keyboard. The modifications made as a result of this study showed the most significant improvements in speed and accuracy.

We tested six independent configurations with varying position and rotations. The error rate, false detection rate, and speed are an averages from the same four participants. As mentioned previously, the six configurations were presented to the users in four independent, random orderings to reduce the effect on the measurements from novelty or fatigue. The first configuration is identical to the one we used in user study 2 and acts as the ground truth. The goal of this study was to find the optimal configuration.

User Study 3			
Config	Error Rate	False Det Rate	Speed
1	0.26	0.21	3:22 min
2	0.29	0.17	4:36 min
3	0.16	0.13	2:39 min
4	0.21	0.29	3:11 min
5	0.31	0.15	3:40 min
6	0.25	0.33	2:56 min

Figure 5: This was a study to find the most ideal configuration for both the head and keyboard. Configuration changes include rotation about the x-axis for the head and transforming the y and z positions of the keyboard. Configuration 3 represents the same configuration that we used in User Study 2.

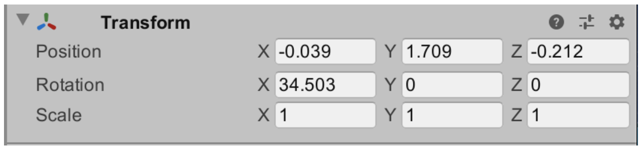
As you can see from table 5, the most ideal configuration we found is Config 3. You can see the specific transformation settings for both the keyboard and the Google Cardboard Rig in images 6 and 7.



Transform

Position	X -0.039	Y 1.579865	Z 0.3396608
Rotation	X -20.538	Y 0	Z 0
Scale	X 0.9	Y 0.9	Z 0.8

Figure 6: These are the ideal transform settings for the keyboard. As you can see the keyboard is offset in the mostly in the y and z direction to position it directly in front of the user's face. The rotation around the x axis allows the keys in the top rows to be farther than the keys at the bottom.



Transform

Position	X -0.039	Y 1.709	Z -0.212
Rotation	X 34.503	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Figure 7: The settings for the cardboard were very similar to that of the original LeapRig. The cardboard is rotated around the x axis to simulate our head orientation. We also translated up the y axis and down the z-axis.

5. Discussion

The latency of our final product was limited only by the Unity Engine itself and the hardware constraints of the computer it is running on. Our current frame rate is capped at 30 fps due to restrictions on the the View Master LCD screen, however this can be easily altered by using a different head-mounted display.

Although users reported some of the usual discomfort associated with VR headsets, including vergence and binocular disparity issues, many found that using the head-mounted display with the keyboard was much easier because they had a better perception of their hands' depth and z-axis location.

Surprisingly, in none of the user studies was the lack of haptic feedback ever brought up as an issue with the VR environment, and after adding the hover light indicators and audio feedback on button press, those changes seem sufficient to simulate the impression of a physical keyboard presence.

Users were most concerned with the difficulty of hitting the letters because the 3-D space requires more precision. As such, it is evident that finding the most ideal orientation for the keyboard was an effective and efficient route for an easier typing experience. Our single user study on the various positional and rotational configurations was a start, but by no means a conclusive or exhaustive effort on the matter. We believe more rigorous testing is required to find a truly optimal keyboard position. Further, we believe that keyboard design studies would prove immensely beneficial, but we did not have the time or resources to undertake such a study.

Another area of particular interest to investigate moving forward is Desktop vs. Head-Mounted Display mode for the Leap Motion. It is our belief that Desktop mode (where the Leap Motion is placed on the desk, facing upwards) would provide a more natural and more ergonomic typing experience, as opposed to the largely vertical experience of the head-mounted display. We attempted to utilize the Desktop mode capabilities of the Leap Motion for our project, but unfortunately the feature is unstable in the Leap Motion Unity package.

Our third user study found that even minor changes to the axis and rotation resulted in significant typing speed and accuracy improvements, so opening up this additional dimension should provide meaningful and significant test results.

Finally, we found that the typing times using VRTyper were not practical over medium to long stretches. We believe there is a valid argument that in-air typing can be useful for short sequences of user input such as typing in a username or password, or a simple command, but it is certainly not ideal for long sequences. To address this issue, we see a potential solution in enabling two different modes in the Unity application: one that allows the user to type in the air, and another that projects the keyboard onto the nearest real-world surface to allow the user to type on something tactile.

This solution carries promise at face-value; indeed, a common use case of such technology might be a VR classroom or VR office space, where we expect users to be seated at a table with their head-mounted display. Further, projecting the keyboard onto the table would alleviate issues faced when interacting with empty space, e.g., moving your hands through the keyboard. Of course, there are other potential avenues to explore in search of an optimal solution for accepting user input of varying length in VR, including, notably, speech recognition/speech-to-text functionality. One potential outcome of further testing and results of VR keyboards, if shown to be non-ideal for medium to long inputs, is the encouragement of research into alternative methods of input, like speech.

6. Conclusion

First, to evaluate the extent to which we achieved our initial projects goals, we, indeed we able to create a virtual reality keyboard with rather low bulk and cost, using only a View Master VR head-mounted display and Leap Motion Controller. With no gloves or controllers, our the VRTyper achieves lower bulk than many existing VR keyboard.

Following on that note, we were able to confirm our initial hypothesis that users would enjoy a glove-less and controller-less experience; however, the naturalness of the typing experience was limited by only being able to utilize Leap Motion's head-mounted mode rather then tracking hand motion from underneath, which would simulate a typing experience more similar to the one we are all familiar with. In the same vein, our results do not support in-air typing as an effective method of gathering user input without significant improvement, as false detection rates and typing speed remain as major obstacles.

The point-and-click nature of the VR keyboard set up, rather than realistic typing with individual fingers able to quickly transition between keys, was identified as the primary limiting factor of the users' typing speed.

Looking towards the future, with the improvements we were able to achieve in light of our significant time and resource constraints, we are hopeful that with further modification and testing, a VR keyboard can become a fully realized solution, or, in the event that it is not deemed feasible for certain length of inputs, serve as the basis for investigating other channels of user interaction.

References

[1] BananaKing. Vr virtual keyboards. `https://www.unrealengine.com/marketplace/en-US/slug/vr-virtual-keyboard`, 2017.

[2] M. Brown. Exploring the magic behind the htc vive controller. *VR Heads*, page 1, 2016.

[3] H. Du, T. Oggier, F. Lustenberger, and E. Charbon. A virtual keyboard based on true-3d optical ranging. In *Proceedings of the British Machine Vision Conference*, volume 1, pages 220–229, 2005.

[4] M. A. Finlayson. *SwissRanger SR-3000 Manual*. MESA Imaging AG.

[5] B. Lang. Analysis of valves lighthouse tracking system reveals accuracy. (1), 2016.

[6] J.-W. Lin, P.-H. Han, J.-Y. Lee, Y.-S. Chen, T.-W. Chang, K.-W. Chen, and Y.-P. Hung. Visualizing the keyboard in virtual reality for enhancing immersive experience. In *ACM SIGGRAPH 2017 Posters*, page 35. ACM, 2017.

[7] J. Nafarrete. Googles daydream labs shows off early vr experiments. *VRScout*, page 1, 2016.

[8] A. J. F. Natoli. Virtual reality keyboard system and method, July 29 2003. US Patent 6,600,480.

[9] J. Ravasz. Punchkeyboard. `https://github.com/rjth/Punchkeyboard`, 2017.

[10] L. M. U. SDK. *Interaction Engine*. Leap Motion.

[11] F. Weichert, D. Bachmann, B. Rudak, and D. Fisseler. Analysis of the accuracy and robustness of the leap motion controller. *Sensors*, 13(5):6380–6393, 2013.

[12] C.-M. Wu, C.-W. Hsu, T.-K. Lee, and S. Smith. A virtual reality keyboard with realistic haptic feedback in a fully immersive virtual environment. *Virtual Reality*, 21(1):19–29, 2017.