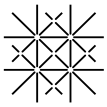

University of Basel

Word-Gesture Typing in Virtual Realty

Bachelor thesis

Databases and Information Systems
Department of Mathematics and Computer Science
Databases and Information Systems
https://dbis.dmi.unibas.ch/

Examiner: Prof. Dr. Heiko Schuldt
Supervisor: Florian Spiess

Philipp Weber
phil.weber@stud.unibas.ch
0000-000-000

20.06.2022

Acknowledgments

So Long, and Thanks for All the Fish. And the template.

Abstract

Text-entry is one of the most common forms of computer-human interaction and indispensable for many tasks such as word processing and some approaches to multimedia retrieval. The conventional keyboards everybody knows have long-established as the main text input method for desktop and laptop computers and even for touchscreen based devices they are very useful. But when it comes to virtual reality (VR) and augmented reality (AR), with today's technology, it lacks of tactile feedback and accurate finger tracking. As a result, text input for VR and AR is still an area of active research.

In recent years, word-gesture typing/ slide-to-type keyboards have been introduced in most major smartphone operating systems. To show the power of such keyboards, we can make a comparison between the best possible performance on a conventional keyboard with the qwerty layout and a word-gesture keyboard with the ATOMIK layout. MacKenzie and Zhang [2] found, that after about 17 hours of practicing, the user of a conventional keyboard with qwerty layout could input about 45 words per minute. On the other hand, Zhai and Kristensson [1] had in their experiment with a word-gesture keyboard with the ATOMIK layout a record input speed of about 52 to 86 words per minute. This shows, that the potential of such word-gesture keyboards is high and one could really write faster after some training. Therefore, we may ask ourselves if this could also be an efficient text input method for VR/AR.

Table of Contents

1 Introduction

1.1 Motivation

To work with vitrivr-VR we need a possibility to input some text into the query text fields. While there already exists a keyboard, it is maybe not the handiest one. The currently implemented one is a conventional keyboard, where a user has to tap on single letters to input single characters. Even though these "query texts" will only consist of a single word or some few words, it still might be exhausting for our arms to input these. The keyboard in vitrivr-VR has a bigger scale than a normal keyboard in reality. We have to move quite a distance with our arms and always move up and down to not accidentally hit a key. With a word-gesture keyboard, the text input could become more comfortable. We would not need to move our arms up and down, we could just move on a flat plane from one key to another. Such a keyboard could also be smaller, because the precision is not as important as it is for the normal keyboard. For example, on a conventional keyboard, if we would tap in the middle of two keys, we cannot really tell which one to take for the input. But with a word-gesture keyboard, where we work with distances and graphs (more on this later), it has not that much of an impact. Therefore, a smaller keyboard is possible, and we do not have to move our arms that much. Hence, it might be less exhausting to write with a word-gesture keyboard.
Other VR applications often do also only use a normal keyboard as text input method. The word-gesture keyboard developed in this thesis will be available as open-source unity package and available for everybody. Therefore, developers of other VR applications may also be using our word-gesture keyboard if they are interested.

1.2 Goals

For this thesis, we have two main goals. The first one is to develop a word-gesture keyboard. This is a keyboard, that more or less might look like a normal one. But instead of tapping on the different keys, words are written with gestures. It has to work with vitrivr-VR and has to be available as open-source. It should also be available as a Unity package, so other developers can use it in their Unity projects as well.
The second goal is to evaluate said keyboard. The evaluation will be conducted according

to current research standards with the usage of the MacKenzie phrase set.

2

Related Work/ Background

In this chapter we introduce the environment the word-gesture keyboard is mainly developed in and developed for, some things about conventional and word-gesture keyboards in general and SHARK2.

2.1 vitrivr-VR and UnityVR

vitrivr[1] is an "open source full stack content-based multimedia retrieval system"[2]. It supports video, image, audio and 3D collections. It also features a very broad set of query paradigms that are supported. vitrivr was developed by the Database and Information Systems group[3] (dbis) of the university of Basel. For our thesis, we use the VR part of vitrivr, namely vitrivr-VR. This is being developed in Unity[4].
Unity is a tool for developers, where one can create projects in 2D, 3D and VR. To a certain degree, Unity is free to use. Developers can provide assets and Unity packages. These can be either free to use or have to be bought. Another developer then can import and use these in their own Unity projects. The main language used in Unity is C#. A developer can write such C# scripts and if needed attach them to objects in a scene. These scripts can control the objects and what they are doing when a user interacts with them or something particular happens.

2.2 Conventional Keyboard

In this thesis, when we talk about conventional keyboard, we do not look at its type of construction, if it's a mechanical keyboard or not. We also do not consider a special layout, when we talk about conventional keyboards. We define the term "conventional keyboard" as the most used keyboard type. The one were a user has to input every single letter by tapping the right key or touching the screen at the right place.

[1] https://vitrivr.org/
[2] https://dbis.dmi.unibas.ch/research/projects/vitrivr-project/
[3] https://dbis.dmi.unibas.ch
[4] https://unity.com

On desktop and laptop computers we normally use such a conventional keyboard. One thing that might be different in some countries is the layout. But that does not change the functionality. Conventional keyboards are also the most used keyboard for phones, tablets and other touchscreen-based devices. The only difference is that we do not press physical keys, but tap on the screen, where a certain key is. These keyboards work really well for text input with the previously mentioned devices. But when it comes to virtual reality (VR) or augmented reality (AR), it seems, that this is not the best method to input text. On reason for this statement is, that right now, it lacks of tactile feedback and accurate finger tracking. While this could be improved during the next years, yet it is not really there. Another reason is the size of such keyboards in VR. We have to tap on the keys with our controllers. If the keys are too close together, it might cause a problem in recognizing which key the user wanted to press. Therefore, there needs to be either bigger keys or bigger spaces between two adjacent keys. This results in a bigger keyboard, which results in more needed movement with the arms. If we have to move our arms a lot to input some text, this can quickly become exhausting.

2.3 Word-Gesture Keyboard

A word-gesture keyboard may look pretty much the same as a conventional keyboard described in the last section, but works quite different. First of all, it does not exist in a hardware version like the conventional keyboard does. It is more like the soft keyboard version, which is a keyboard displayed on a screen, like the ones used when typing text on a phone.

Independent of the details of the implementation, every word-gesture keyboard (also called slide-to-type keyboard) works with gestures. That means, instead of tapping on single keys, the user has to draw one line or a shape on the keyboard. This will then be evaluated by an algorithm, that determines the closest word, the one with the most similar shape seen from different aspects, from a lexicon. For example, to input the word "science", the user has to put the finger on the screen, where the "s-key" is displayed. Then they have to move, with the finger still on the screen, to the respective adjacent key with the correct character. At the end, the user has to take away their finger from the screen at the "e-key". If the gesture was more or less good, the algorithm behind should now be able to calculate, that "science" is the word, the user intended to write. But if the gesture is done bad, it can happen, that the wrong word is being calculated.

2.3.1 SHARK2

SHARK2 is a "large vocabulary shorthand writing system for pen-based computers" [1] invented by Shumin Zhai and Per-Ola Kristensson. It can compare the user inputted graph with a perfect graph of any word in a given lexicon. A perfect graph is the graph, that is produced, if we start from the center of the word's first letter on the keyboard. Then we draw a straight line to the center of the next letter of the word and so on, until we reach the last letter.

The SHARK2 system needs a lexicon with words and all their perfect graphs stored. To get the most probable word from the lexicon, the user intended to write, it uses a multi-channel recognition system. Each channel alone from the system developed by Zhai and Kristensson [1] does not necessarily have enough power, but all the channels together can detect the right word. The two core channels are the shape and location recognizers.
First of all, $SHARK^2$ uses template pruning. It compares the start and end positions of the perfect graph of each word in the lexicon with the normalized input gesture from the user. If one of these two distances is bigger than a given threshold, the checked word will be discarded and not further considered. With normalized, the authors mean normalized in shape and location.
The next step is to apply the shape recognizer. It compares the shapes of the perfect graph for every word in the lexicon and the user inputted graph. For this, an amount of N sampling points has to be calculated for every graph. These N points need to be equidistant. Then they have to be normalized in scale and location. This means, that the graphs are all normalized by scaling the largest side of the graph's bounding box to a predetermined length L:

$$s = L/max(W, H) \tag{2.1}$$

W and H are the width and height of the graph's bounding box. All points' positions have to be divided by s to get the normalized points' positions. After that, the middle point of every graph has to be set to the point (0,0). Now the distance between the user inputted graph and every word's perfect graph, that is in the lexicon, has to be calculated. To do so, we use the following formula:

$$x_s = \frac{1}{N} \sum_{i=1}^{N} \|u_i - t_i\|_2 \tag{2.2}$$

where u_i is the ith point of the user input graph and t_i the ith point of a word's graph. This is the so-called proportional shape matchimg distance [1]. Now, one could think, that this is enough and with the application of the template pruning and shape channel recognition, the word is perfectly determined. This is not the case. The authors stated, that words can have a similar or even same shape as other words. They call these word pairs "confusion pairs". They found out, that for example on an ATOMIK layout with a lexicon of 20'000 words, there were 1117 pairs of words that have an identical graph, if the starting and ending positions are not considered. If these are also considered with the shape, there is still a total of 537 confusion pairs.
To avoid these, the authors were using a second channel, not for the shape, but for the location. For the following formulas and calculations, the normalization of the graphs in not needed anymore. As the name states, it is more about the location, where the graph lies in a coordinate system, than about its shape.
They use an algorithm that computes the distance of the user inputted graph u to the perfect graph t of every word in the lexicon. The location channel distance is defined as:

$$x_L = \sum_{i=1}^{N} \alpha(i)\delta(i) \tag{2.3}$$

where N is the number of points used to sample a graph. $\alpha(i)$ with $i \in (1, N)$ are weights for the different point-to-point distances, such that $\sum_{i=1}^{N} \alpha(i) = 1$. $\alpha(i)$ can be valued in various ways. For $SHARK^2$ the authors used a function, that gives the lowest weight to the middle point-to-point distance. For the other point-to-point distances, the weight increases linearly towards the two ends. $\delta(i)$ is defined through following formula:

$$\delta(i) = \begin{cases} 0, & D(u,t) = 0 \wedge D(t,u) = 0 \\ \|u_i - t_i\|_2, & \text{otherwise} \end{cases} \tag{2.4}$$

where u_i is the i-th point of u and t_i the i-th point of t. D is defined as:

$$D(p,q) = \sum_{i=1}^{N} \max(d(p_i, q) - r, 0) \tag{2.5}$$

r is the radius of a key and d is defined as:

$$d(p_i, q) = \min(\|p_i - q_1\|_2, \|p_i - q_2\|_2, \ldots, \|p_i - q_N\|_2) \tag{2.6}$$

For all these formulas N has the same definition as for formula 2.3. The "trick" the authors use these formulas for is pretty simple. They state, that they form something like an "invisible" tunnel of one key width that contains all keys used to write a certain word. A perfect distance score of zero is given, when all the sampled points of the user inputted graph lie within the tunnel of t. If this is not the case, the distance score for t with respect to the user inputted graph is set to the sum of the spatial point-to-point distances.
With the two distances x_S and x_L, the most probable word, the user intended to write, can be calculated. The authors assume, that the distance from a user inputted graph to the perfect graph of a word follows a Gaussian distribution. This means, if the user inputted gesture has distance x to a perfect graph of word y, the probability, that y is the intended word can be calculated using the Gaussian probability density function:

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right] \tag{2.7}$$

To do so, the authors used the following three formulas:
The final result c(w) for every word w that met the requirements tells the probability that w is the word, the user intended to write with his/her gesture.
The authors say, that the user can draw a graph either on visual guidance from the keyboard (looking for the next letter of a word on the keyboard) or recall from memory. A graph drawn by visual guidance results in a higher location distance score than a graph drawn from memory recall. If a user draws a graph by memory recall the location distance score will be poor and the focus lays more on the shape. Therefore, they suggest a dynamic weighting of the two channels, that is to adjust the weighting of the channels according to the time needed to draw the graph. In general, graphs drawn by memory recall are faster than visual guided ones. Hence, the gesture completion time should tell, how heavy the location channel should be weighted in the final selection. The time to complete a graph for a word obviously depends on its length and complexity. The authors then use Fitts' law (TODO: EITHER FORMULA OR REFERENCE TO IT) to calculate the normative writing time for a graph

of a word. They use this result together with the actual graph production time to modify δ in (TODO: REFERENCE TO FORMULA THEY SUB DELTA).
The authors achieved quite satisfactory performance with the two channels, but there still might be conflicting words. To prevent these, the authors suggest to also use language information. For SHARK2 smoothed bigrams are used as the language model. It is then used to rearrange the N-best list of words received before.

3
Implementation

In this section, we will introduce how we implemented a word-gesture keyboard using Unity and a python script and how we used SHARK2 for our algorithm.

3.1 Word Graph Generator

As mentioned in the related-work SHARK2 part TODO: REFERENCE TO MENTIONED SECTION, to make SHARK2 work, the perfect graphs for all the words in a lexicon are needed. Therefore, we needed a script, that produces or overwrites a file for every available keyboard layout and writes the sampled points of every graph in it. Such a file contains per line a word, then a certain number of points from the word's perfect graph followed by the same points, but normalized (normalized as mentioned in the related-work SHARK2 section) TODO: REFERENCE TO MENTIONED SECTION.
To run the script, the user has to provide the name of the layout that he/she wants to create the perfect graphs for. Additionally, he/she has also to write the name of the text file containing all the words (lexicon). The script then either creates a new file named "sokgraph_*layout*.txt" or if already a file with this name exists, it deletes its content to write new in it. Then it fills the file line by line as mentioned above.
The file can only be executed for one layout a time. Hence, if there are more available layouts for our word-gesture keyboard, the user has to run the script several times.

3.2 Used Algorithm

For our word-gesture keyboard we used a weaker version of SHARK2. This means, we do also work with two channels, a location channel and a shape channel. The shape channel is to calculate the deviation from the user inputted graph and a perfect graph from a word in the sense of distance with respect to their shape, the location channel is for the same thing, but not for the shape, but rather the position. When looking at the shape, we have to normalize the graphs in a specific way, so the position, where they exactly lie in a coordinate system does not matter. When looking at the position, we look at the graphs as they are, without normalizing or changing anything. As in the SHARK2 system we also use the start

and end positions of the graphs as pruning method. The difference is, that for SHARK2, the authors chose to use normalize all the graphs in scale and translation before comparing. We do not normalize the graphs, but just look at the start and end positions of a user input graph and a word's graph. Another thing we have almost implemented the same as it is in SHARK2 is δ. For the shape and location channel integration, the used δ in SHARK2 is equal to the radius of one key. We do the same for the location channel (in the integration part), but we do not use the same δ for the shape channel (in the integration part). For this, we take a δ that equals the radius of a normalized key. That means, a small graph will have a bigger delta than a big graph, because we do normalize the graphs and a small graph gets stretched by it, whereby a big graph gets drawn together. TODO: REVISE.
However, we do currently not use any language information nor dynamic channel weighting by gesturing speed.

3.3 Functions

The most important, but also most basic function our word-gesture keyboard provides, is the writing of words with gestures. A user can press the trigger button of a VR controller inside the keyboards hitbox and start making a gesture on it. The user will see a TODO: SEE COLOR. red/purple line that is drawn on the keyboard where he/she moves. This helps the user to keep track of the trace he/she drew. When the user wants to finish the gesture, he/she needs to release the trigger of the controller. At this moment, our program starts to evaluate the 5 words with the highest accordance to the user inputted graph. The one with the highest accordance will be written into the text field. The other 4 are displayed at the keyboard TODO: CITE TO PICTURE WITH RECOMMENDATIONS., such that the user also can choose between these. When the user chooses one of these 4 words, the word that has been written into the text field before is getting replaced by the user's chosen word and the key, where the chosen word was written will then display the replaced word.
Because the whole system works with a lexicon full of words and only these can be written as gestures, there will also be word, the user wants to write, that are not yet in said lexicon. For this case, we implemented a function such that the user can add new words. He/She can access via an options button marked with a black gear right above the keyboard the "add word" button. When this button is pressed another button appears, the "add word to dict" button TODO: REVISE. When this button is pressed, the word (it does not need to be one, that really exists) displayed above the keyboard, will be added to the lexicon text file. Additionally, for every available layout, the corresponding graph for the newly added word will be added to the corresponding "sokgraph_*layout*.txt" text file. One additional thing we implemented is, that the word's graph only gets appended in the text file, if this word can be written with the layout the file corresponds to. For example, if a user wants to add the word "öffentlich", but he/she made a layout without the letter "ö", this word could never be written with said layout, hence it would be unnecessary to have this word in said "sokgraph_textitlayout.txt" text file.
Another function that is necessary for the previous function is the possibility to input single letters. If a word does not yet exist in the lexicon or layout text file, then it can-

not be written with a gesture. Hence, we needed a way to input single letters. Fortunately with a word-gesture keyboard this almost works without additional work. If the single letters are as graphs in the layout files, TODO: DEFINE LAYOUT FILES AS SOKRAPH_LAYOUT.TXT SOMEWHERE. the user is more or less able to write single letter with a gesture (just a click on the right key). But there might be a little inconvenience. This is caused by the fact, that we work with distances. The distance from one letter to another is not too big. And if for example the user wants to write an "e" but presses the key with the "e" on it on its left side and not perfectly in the middle, our system would also evaluate that aside from "e", also "w" and "we" are words, the user might have intended to write (on a conventional qwertz or qwerty layout). To get rid of this, our system checks, if the user inputted graph's bounded box is smaller than TODO: HOW MUCH SMALLER?. it recognizes, that the user wants to write a single letter, and then takes the best match. To get back to the example, "we" would be discarded and "e" would get a higher score than "w", because of a smaller location channel distance. Therefore, the written "word" in this case would be "e".

Another function to help the user is the creation of own layouts. A layout text file exists, that contains all available layouts. The user can as many new layouts as he/she wants to. To create a new one, the user has to give the layout he/she wants to create a name and on the following lines write the characters in an order, that he/she wants to have on the keyboard. All the unicode characters should be working, but two. In the current implementation, one whitespace is used to declare the position of the spacebar and the "¡" character is used for the backspace key. This file gets read at the start of the program, so it cannot be edited while the program is running, or to be precise, the changes will not be recognized during runtime. One smaller thing we implemented is, that at the start of the program TODO: START OF PROGRAM OR MORE LIKE APPEaRANCE OF KEYBOARD?. all characters used in the layout not yet in the lexicon text file and "sokgraph_*layout*.txt" are being added. Without doing this, the user may not be able to input any single special character with his/her newly created keyboard, because the system simply does not find it in the layout files. The user is during the runtim able to switch between available layouts whenever he/she wants to.

To end the implementation part, we will talk shortly about two small functions. One is, that the keyboard is grabable. This means, the user can grab the keyboard and move it around in the room. We were able to do this thanks to script that already existed in vitrivr and did not have to implement anything on our own.

The last small function is the ability to change the size of the keyboard. This can help the user writing words, depending on how he/she wants to move his/her arm and therefore put the keyboard a bit closer or further away.

—"draw" words with gestures/put single letters —create own layout —change size —choose between best 5 words —keyboard is movable —chose between available layouts all the time possible —add new words

4
Evaluation

In this section we want to talk about the evaluation as a whole. We want to look at the phrases we took, how we conducted it, the results observed and shortly discuss what all of this means.

4.1 MacKenzie Phrase Set

One precondition for the evaluation was to use the MacKenzie Phrase Set TODO: CITE TO PAPER. Basically, this is just a set of 500 phrases. According to the paper, such a phrase set should use phrases of moderate length, that are easy to remember and representative for the target language. These phrases do not contain any punctuation. Some of them use uppercase characters, but the authors mention, that participants can also be instructed to ignore the case of the characters. The complete MacKenzie Phrase can be downloaded at http://www.yorku.ca/mack/PhraseSets.zip.
Some statistics for the whole phrase set, also found in the original paper TODO: CITE PAPER: The MacKenzie phrase set consists of 500 phrases, that have a minimum length of 16, a maximum length of 43 and average length of 28.61 characters. On the whole, 2712 words were used, which consist of 1163 unique words. A phrase consists of a minimum of 1, a maximum of 13 and on average of 4.46 words.

4.2 Task of the Participants

The task of the participants was to copy 15 "random" phrases from the MacKenzie phrase set. They are not really random, but adjacent phrases in the downloadable MacKenzie Phrase Set (http://www.yorku.ca/mack/PhraseSets.zip). As they are not in a specific, e.g. alphabetic order, we decided to do it like this.
The participants could see two text fields. On the top was the phrase to copy, on the bottom the words/phrase they wrote. If the given phrase matched the user inputted phrase, a sound would sound, such that the participants knew when they finished one specific phrase. After that, a new phrase would appear until 15 phrases were correctly inputted.
After this first step, in the second step, we shortly explained two other functions of the

keyboard. First the scaling buttons and then the function to add a new word. This is important, because we wanted to know if they found these functions useful and well implemented.
The last step of the evaluation was to fill a questionnaire TODO: ALS ANHANG BEIFÜGEN.

4.3 Carry-out

The VR system we used are the HTC vive TODO: GET RIGHT NAME AND STUFF OF VR STUFF. To find participants, we wrote an email to students from our university, and asked family members and friends not involved in the process of making this bachelor's thesis. All in all, TODO: HOW MANY PEOPLES? people got in touch with us and participated at our evaluation. Every participant got the same explanation. We told them that the keyboard is movable, if they are close enough to the keyboard (the color then gets a bit brighter) and press the controller's grip button. To write, they do also have to be in the hitbox of the keyboard but not pressing the grip button, but the trigger button. We also told them, that if they do a full gesture and a word longer than one character is written, a space is automatically put behind the word. If they put in a single character, by clicking on that key of this character, no space is put, and they have to do it their own. This is particularly important for the words "I" and "a". We told also told them, that if they made a gesture and a word was written, there may be one to four other choosable words they could pick, if the word displayed as first is the wrong one. We also told the participants, that we have enough time and that they should not hurry, but rather look, that the inputted words are correct. Because if they are not correct, they have to use the backspace a lot of times.

4.4 Results

4.5 Discussion

5 Conclusion

This is the body of the thesis.

5.1 Results Discussion

5.2 Future Work

Bibliography

[1] Per Ola Kristensson and Shumin Zhai. Shark2: a large vocabulary shorthand writing system for pen-based computers. In *UIST '04*, pages 43—52, 2004.

[2] I. Scott MacKenzie and Shawn X. Zhang. The design and evaluation of a high-performance soft keyboard. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 25—-31, 1999.

A

Appendix

University of Basel

Faculty of Science

Declaration on Scientific Integrity

(including a Declaration on Plagiarism and Fraud)
Translation from German original

Title of Thesis:

Name Assesor: _____

Name Student: _____

Matriculation No.: _____

With my signature I declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged. I have mentioned all source materials used and have cited these in accordance with recognised scientific rules.

Place, Date: _____ Student: _____

Will this work be published?

☐ No

☐ Yes. With my signature I confirm that I agree to a publication of the work (print/digital) in the library, on the research database of the University of Basel and/or on the document server of the department. Likewise, I agree to the bibliographic reference in the catalog SLSP (Swiss Library Service Platform). (cross out as applicable)

Publication as of: _____

Place, Date: _____ Student: _____

Place, Date: _____ Assessor: _____

Please enclose a completed and signed copy of this declaration in your Bachelor's or Master's thesis .