Improving everyday computing tasks with head-mounted displays

James Walker
Michigan Technological University, jwwalker@mtu.edu

IMPROVING EVERYDAY COMPUTING TASKS WITH HEAD-MOUNTED DISPLAYS

By

James Walker

A DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

In Computer Science

MICHIGAN TECHNOLOGICAL UNIVERSITY

2017

This dissertation has been approved in partial fulfillment of the requirements for the Degree of DOCTOR OF PHILOSOPHY in Computer Science.

Department of Computer Science

Dissertation Advisor:	*Scott Kuhl*
Committee Member:	*Keith Vertanen*
Committee Member:	*Robert Pastel*
Committee Member:	*Myounghoon Jeon*
Committee Member:	*Peter Willemsen*
Department Chair:	*Min Song*

Table of Contents

List of Figures

List of Tables

Preface

Chapter 2 incorporates material from the publications *Decoder-Assisted Typing Using an HMD and a Physical Keyboard* [53] and *Efficient Typing on a Visually Occluded Physical Keyboard* [54] by Walker, Li, Kuhl, and Vertanen. Walker, Kuhl, and Vertanen co-wrote the paper; Walker developed the system by building on VelociTap source code provided by Vertanen; Walker and Li performed the user studies; and Vertanen was in charge of data analysis.

Chapter 4 incorporates material from the publication *Minification affects verbal- and action-based distance judgments differently in head-mounted displays* by Zhang, Nordman, Walker, and Kuhl [67]. Zhang, Walker, and Kuhl co-wrote the paper; Zhang developed the experimental software; and Zhang, Nordman, and Walker performed the user study.

Appendix A incorporates material from the publications *FlowTour: An automatic guide for exploring internal flow features* by Ma, Walker, Wang, Kuhl, and Shene, *Constructing the immersive interactive sonification platform (iISoP)* by Jeon, Smith, Walker, and Kuhl, *Interactive sonification markup language (ISML) for efficient motion-sounding mappings* by Walker, Smith, and Jeon, *Technologies expand aesthetic dimensions: Visualization and sonification of embodied Penwald drawings* by Jeon, Landry, Ryan, and Walker, and *Enabling Affordable Industrial Robotics Education through Simulation* by Kuhl, Sergeyev, Walker, Lakshmikanth, Highum, Alaraje, Zhang, and Kinney. Walker co-wrote all of these papers and also provided software development, data gathering, or data analysis for them.

The error bars in all figures indicate the standard error. Participants provided informed consent as part of the procedure for all of the studies described in this work.

Acknowledgments

Thank you to my advisor, Dr. Scott Kuhl, for bringing me the opportunity to do my doctoral research in the cutting-edge field of virtual reality, for accepting me into the VR research group, for providing me with multiple funding opportunities, and for being an amiable mentor. I was very lucky to have an advisor who was a good fit for my personality and working style for my doctoral research.

Thank you also to my committee members, especially: Dr. Keith Vertanen for providing invaluable assistance during the last leg of my research, particularly with the VelociTap decoder, by providing excellent data analysis, and helping to get the text entry research into publishable shape; Dr. Myounghoon “Philart” Jeon for the opportunity to do software development for the artistically-oriented iISoP system; and Dr. Robert Pastel for valuable guidance on performing the user experience survey for FOW.

Thank you to the other professors who helped make my time at Michigan Tech enjoyable, including (but not limited to) Dr. Charles Wallace, Dr. Soner Onder, Dr. Nilufer Onder, and Dr. Saeid Nooshabadi. Special thanks to Dr. Jean Mayo, Dr. Ching-Kuang Shene, and Dr. Steven Carr for accepting me on the VACCS project.

Thank you to my friends and research colleagues Bochao Li and (now Dr.) Ruimin Zhang. Doing research alongside likable peers makes it more fun than going it alone.

Last but not least, thank you to all my other friends for making life fun and thank you to my family for all the love and support. My accomplishments didn’t come out of a vacuum and they wouldn’t have been possible without you.

Abstract

The proliferation of consumer-affordable head-mounted displays (HMDs) has brought a rash of entertainment applications for this burgeoning technology, but relatively little research has been devoted to exploring its potential home and office productivity applications. Can the unique characteristics of HMDs be leveraged to improve users' ability to perform everyday computing tasks? My work strives to explore this question.

One significant obstacle to using HMDs for everyday tasks is the fact that the real world is occluded while wearing them. Physical keyboards remain the most performant devices for text input, yet using a physical keyboard is difficult when the user can't see it. I developed a system for aiding users typing on physical keyboards while wearing HMDs and performed a user study demonstrating the efficacy of my system.

Building on this foundation, I developed a window manager optimized for use with HMDs and conducted a user survey to gather feedback. This survey provided evidence that HMD-optimized window managers can provide advantages that are difficult or impossible to achieve with standard desktop monitors. Participants also provided suggestions for improvements and extensions to future versions of this window manager.

I explored the issue of distance compression, wherein users tend to underestimate distances in virtual environments relative to the real world, which could be problematic

for window managers or other productivity applications seeking to leverage the depth dimension through stereoscopy. I also investigated a mitigation technique for distance compression called minification. I conducted multiple user studies, providing evidence that minification makes users' distance judgments in HMDs more accurate without causing detrimental perceptual side effects. This work also provided some valuable insight into the human perceptual system.

Taken together, this work represents valuable steps toward leveraging HMDs for everyday home and office productivity applications. I developed functioning software for this purpose, demonstrated its efficacy through multiple user studies, and also gathered feedback for future directions by having participants use this software in simulated productivity tasks.

Chapter 1: Introduction

Since the advent of computing, the prospect of virtual reality—interacting with an immersive, computer-generated environment that seems as real and engaging as reality—has captured many imaginations. Not until recently, with the advent of inexpensive, consumer-available head-mounted displays (HMDs), has virtual reality experienced the opportunity to move out of research laboratories and into people's homes.

Yet the possibilities of this technology arguably remain underutilized. Laboratory research into virtual reality has focused on fundamental issues like navigating large virtual environments in physically limited spaces and fields such as training, virtual prototyping, or therapy—esoteric applications for the average end-user. Commercial applications have focused almost exclusively on entertainment, such as video games and 3D films. Although consumers value entertainment, most people use their computers for more than just playing video games. Could virtual reality be used to enhance everyday computing tasks?

The heart of this work strives to explore this question. First, I address fundamental obstacles to the use of virtual reality for everyday tasks. In particular, the fact that most models of HMDs occlude the real world poses an impediment to interacting with physical peripherals and the surrounding environment. I describe the development of a system to aid users typing on physical keyboards while wearing HMDs and a series of experiments evaluating the effectiveness of my system.

Next, I build on this work to develop a prototype of a system capable of supporting everyday computing tasks—a window manager optimized for use with HMDs. I describe the development of this window manager and a user experience survey that I conducted to evaluate the software and gather user feedback on future directions for the development of such systems.

After that, I return to the issue of obstacles to HMD usability; specifically, the well-documented phenomenon of distance compression, in which HMD users perceive objects in the virtual environment to be closer than the developer intends. I describe a series of studies exploring minification, a graphical rendering technique to compensate for distance compression.

I have also conducted a significant body of research which does not directly relate to issues of HMD usability. This includes a user study of a flow visualization program; the development of a software library for distributed graphics rendering; iISoP, the immersive Interactive Sonification Platform; and RoboRun, a robotics simulation program for educational purposes. For completeness, I describe each of these research projects.

Chapter 2: Typing on an Invisible Keyboard

2.1. Introduction

Traditional input devices, such as QWERTY keyboards and mice are a popular and efficient way to interact with computers. However, using these devices with HMDs is challenging because the HMD prevents users from seeing the devices. Can this obstacle be overcome, or at least mitigated, so that users are better able to utilize physical keyboards in conjunction with an HMD?

To investigate this, I designed a system that employs two key features to aid users in using a keyboard while wearing an HMD. The first feature is a touchscreen decoder modified for use with physical keyboards that can provide text-correcting capabilities far beyond those of a typical spell checker. The second feature is a graphical keyboard displayed on the screen that lights up each key whenever the user presses the corresponding key on the physical keyboard. I then conducted a pilot study [53] followed by a full study [54] in order to evaluate the effectiveness of this system.

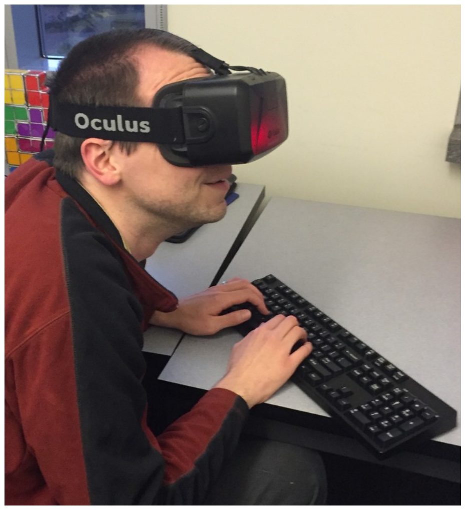


Figure 2.1. Typing on a physical keyboard while vision is occluded by an HMD potentially poses a significant usability impediment.

2.2. Background and Related Work

Relatively little work has implemented or compared text entry systems for HMDs. Bowman et al. [3] compared several methods, including a one-handed chord keyboard, speech recognition using a human instead of software, and a virtual keyboard controlled by a tablet and pen. They found that none of these approaches produced high levels of performance or usability.

The augmented reality system ARKB displays a holographic keyboard on a see-through HMD [31]. ARKB tracks a user's hand position to monitor when a user's fingers touch the keyboard. A more recent example is the Microsoft HoloLens keyboard in which a user's head position controls a pointer on a holographic keyboard with keys selected via a hand gesture. Yi et al. [66] developed Atk, a system that uses 3D hand tracking to enable mid-air typing. Users were able to type in mid-air with high accuracy, but at slower rates than what is typically achievable with a physical keyboard.

Another way to facilitate real-world interaction while wearing an HMD is mixed reality. In mixed reality, portions of the real world are superimposed over the virtual environment. McGill et al. [35] found that users' unassisted typing performance while wearing HMDs was significantly reduced, whereas blending real and virtual reality brought performance closer to baseline. In a study by Budhiraja et al. [5], users took drinks from a cup while interacting with a virtual environment. Users reported they preferred mixed reality solutions compared to removing the HMD to interact with the real world.

Although users have shown a positive response to mixed reality, such systems add substantial software and hardware complexity, and not all users might have the budget or space for external cameras necessary to implement mixed reality. In addition, poor image quality or occlusion by the user’s own body can hamper effective feedback. Systems which rely exclusively on gestures, such as mid-air typing, have to date failed to offer input speed comparable to what is possible with a physical keyboard. Mid-air typing can also be surprisingly fatiguing even after brief periods of use. For these reasons, I wanted to explore developing a system which would allow users to continue utilizing their physical input peripherals without relying on extra hardware. To accomplish this, I developed a typing assistant that utilizes a virtual keyboard to help users type on a keyboard without being able to see it, and a state-of-the-art decoder to correct user input even in the face of substantial errors.

The decoder component of my system was built on the VelociTap touchscreen decoder [50]. VelociTap takes a sequence of noisy touch locations and searches for the most probable sentence given those touches. The decoder has also been used to research eyes-free touchscreen input [51].

2.3. Pilot Study

2.3.1. System Implementation

My prototype system consisted of three components:

1. A keyboard client responsible for randomly selecting phrases to be typed, logging users' keystrokes, and sending that timestamped data to the decoder.
2. The decoding server which forwarded phrases and keystrokes to the display server, computed words per minute (wpm) and accuracy, and ran the decoder.
3. A display server that accepted phrases and keystrokes and displayed them to the user.

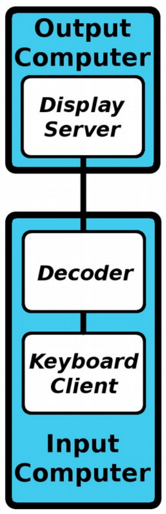


Figure 2.2. The main components of my system.

Each component was capable of being run on separate hardware or on the same piece of hardware. For my prototype system, the keyboard client and decoder were run on one computer while the display server ran on another. Each software component communicated with the others using TCP with Nagle's algorithm disabled in order to prevent transmission delays.

2.3.2. Recognition Details

To perform automatic correction of users' noisy keyboard typing, I modified the VelociTap touchscreen decoder [50]. Normally, VelociTap searches for the most probable sentence given a time-ordered sequence of x- and y-locations recorded by a touchscreen sensor. VelociTap's keyboard model uses two-dimensional Gaussians centered on each key of the onscreen virtual keyboard. In this work, I didn't have a virtual keyboard, I had a real one. I measured a physical keyboard and created a keyboard map based on its physical size and locations measured in millimeters.

During typing, each key down event was mapped to the center x- and y-position of that key on the physical keyboard. I then simulated a touchscreen tap on this center position. This allowed VelociTap to create a probability distribution over all possible keys with keys closer to the key actually pressed having higher probability.

In addition to the keyboard model, VelociTap also uses a letter and word language model. It was trained using a 12-gram letter language model (2.2 GB on disk) and a 4-gram word language model (3.8 GB on disk) on billions of words of data from Twitter, social media, blog, and movie subtitles.

2.3.3. Study

For my pilot study, I had five participants in three within-subject conditions:

- VISIBLE—Participants typed using a fully visible keyboard and a desktop monitor.
- OCCLUDED—Participants typed using a keyboard which was occluded from the user’s vision and a desktop monitor. The keyboard was occluded by placing it underneath a cardboard container which had enough space inside to hold the keyboard and the participant’s hands comfortably, while simultaneously preventing the participant from being able to see the keyboard while typing.
- HMD—Participants typed using a keyboard while wearing an Oculus Rift DK2 head-mounted display.

Participants were seated at a desk in a quiet office and typed on a Dell SK-8115 keyboard. In the conditions where the participant did not wear an HMD, visual feedback was on a 24-inch LCD monitor (1920 × 1080 resolution) positioned about two feet in front of the participant. In the HMD condition, participants used an Oculus Rift DK2 HMD (resolution of 960 × 1080 per eye). I trained participants how to adjust the HMD.

In each condition, participants were shown 30 short, memorable phrases from the Enron mobile data set [49]. Phrases contained only the letters A–Z and apostrophe. Participants were told to memorize each phrase before starting to type. As soon participants began typing, the phrase disappeared and was replaced by the literal characters typed. After typing each phrase, participants submitted the phrase for decoding by performing a long keypress by holding down any key for at least 300 milliseconds. After each entry, participants were shown the entry rate and error rate for their last entry. Participants moved to the next phrase via another long keypress.

2.3.4. Results and Discussion

Table 2.1. Mean and standard deviation of participant entry rates and character error rates (before and after automatic correction).

Condition	Entry rate (wpm)	Before CER (%)	After CER (%)
VISIBLE	57.5 ± 18.2	1.8 ± 2.0	1.4 ± 1.7
OCCLUDED	47.7 ± 26.6	6.8 ± 7.3	3.0 ± 2.8
HMD	39.0 ± 14.8	7.0 ± 5.5	3.5 ± 1.6

Error rate before and after automatic correction is reported using character error rate (CER). CER is the number of character insertions, substitutions, and deletions required to transform the entered text into the reference text divided by the number of characters in the reference (times 100).

Entry rate is reported in words per minute (wpm), with a word being defined as five characters including spaces. My entry rate includes the time required for the long press to finish a sentence, the time to send the keystroke data to the server, the time to perform the recognition, and the time to display the result.

As shown in Table 2.1, the fastest and most accurate entry was obtained when the keyboard was fully visible. Although all of my participants reported to be touch typists, entry rate slowed and error rate rose substantially once the keyboard was occluded. The complete occlusion of vision by wearing an HMD even further impacted speed and accuracy. Comparing the before and after error rates, it is evident that the sentence-based decoding approach successfully corrected about half the errors when the keyboard was covered or participants were wearing the HMD.

I was curious if even more accurate results might be possible if more computation time was spent on decoding. I pooled all the HMD data from the participants and ran offline recognition experiments. VelociTap has a beam parameter controlling its tradeoff between speed and accuracy. With the beam used in the study, the average sentence error rate was 7.77% (0.04 s per decode). Doubling the beam substantially lowered the error rate to 6.30% (0.26 s per decode). Tripling the beam offered only a small additional reduction of error rate to 6.27% (1.63 s per decode).

Based on past experience with VelociTap and touchscreen typing data, it is not typical for additional computation time to yield such accuracy improvements. I conjectured that for physical keyboards there are common error types (e.g. transpositions) that were not explicitly modeled by VelociTap. This suggested that adding additional features to the decoder might offer improved accuracy without widening the search beam.

These results tentatively suggested that even experienced touch-typists may experience reduced speed and accuracy when typing blind, and even more so when using an HMD. This pilot study also provided preliminary evidence suggesting that the decoder could significantly improve users' typing capabilities under these conditions. This provided a solid foundation on which to further expand my system and perform a larger study to more thoroughly evaluate its capabilities.

2.4. Main Study: Decoder + Virtual Keyboard

2.4.1. System Implementation

Based on the results from my pilot study, I implemented two improvements to my text entry system. One of these improvements was the addition of a virtual keyboard to the screen. This virtual keyboard provided high-contrast visual feedback about which keys were being pressed and could be made arbitrarily large. When the keyboard client detected a keypress on the physical keyboard, it forwarded it to the display server which lit up the corresponding key on the virtual keyboard. The glow gradually faded to black over half a second. The gradual fading allowed users to see not only the last key hit, but other recently hit keys (see Figure 2.3). The virtual keyboard had labels for the letters, apostrophe, and comma. Other unlabeled key outlines (e.g. number keys, shift keys) were also shown and these keys lit up if pressed.

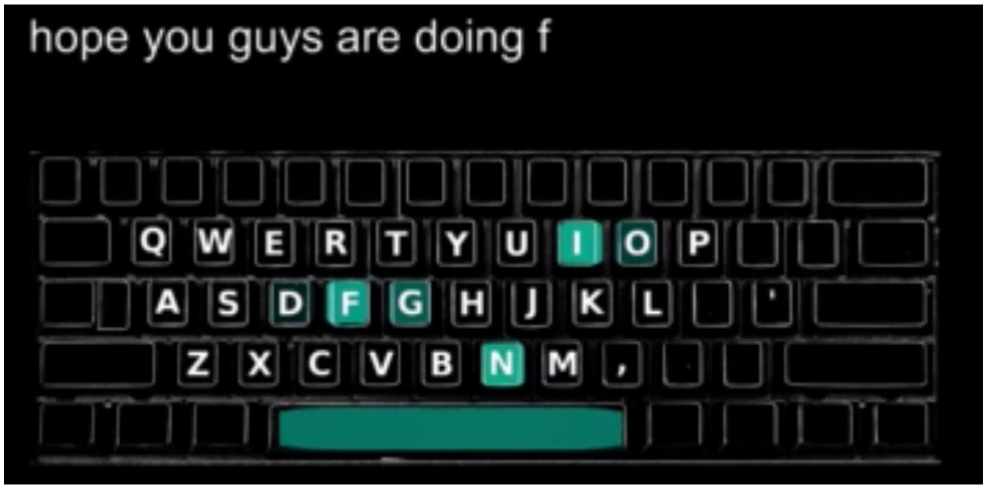


Figure 2.3. The virtual keyboard assistant as shown in the HMD.

The other improvement was to further modify VelociTap for decoding input from physical keyboards. I investigated several potential modifications for this purpose, including modeling left/right hand offsets, discrete probability distributions based on key locations, and modeling multiple adjacent key strikes. However, none of these modifications yielded consistent accuracy improvements.

One modification that did yield measurable results was explicit modeling of transposition, which I had conjectured was a common error type in bimanual keyboard typing. I modified VelociTap to explicitly model transposition by allowing adjacent observations to be swapped instead of requiring a multi-step process of deleting a character, inserting a new one, deleting the next character, and inserting another new one.

Swapping observations incurred a new transposition penalty. All parameters of VelociTap were optimized with respect to data obtained from my pilot study and from data that I recorded myself.

2.4.2. Study

I designed a follow-up study to test two hypotheses: 1) When users cannot see the keyboard, the virtual keyboard with live feedback improves typing performance; and 2) users' typing performance would be worse while wearing an HMD than while the keyboard was merely occluded. To test this second hypothesis, I once again included non-HMD conditions in which I placed a physical cover over the top of the keyboard.

My study had two independent variables, whether the virtual keyboard was shown, and whether visual feedback was via an HMD or a desktop monitor. This resulted in four within-subject conditions. In the Desktop and DesktopAssistant conditions, visual feedback was from the same 24-inch LCD monitor used in the pilot study. In the HMD and HMDAssistant conditions, visual feedback was from the same DK2 HMD used in the pilot study. The virtual keyboard assistant was shown only in the DesktopAssistant and HMDAssistant conditions.

I recruited 24 participants via convenience sampling (7 female, 19 touch typists, ages 19–28). Participants received either course credit or $10. The study took place in the same location, using the same keyboard and other hardware, as the pilot study.

Participants first completed a practice session where they typed 20 phrases. The goal of the practice phase was to allow participants to gain experience with the occluder that covered the keyboard and their hands, to gain experience wearing and looking through the HMD, and to become familiar with the virtual keyboard assistant. After this practice session, participants completed the conditions in counterbalanced order.

In all conditions, participants were shown 20 random memorable phrases from the Enron mobile data set [49]. All other aspects of the procedure were the same as the pilot study except for the different conditions and the presence of the virtual keyboard assistant in two of the conditions. If the virtual keyboard was enabled, it was shown at the bottom of the screen.

2.4.3. Results and Discussion

An unanticipated problem with my procedure was that some participants prematurely typed a long keypress. I believe this happened because many participants used the spacebar for their long keypresses at the end of each phrase. Another long press was required before the next phrase appeared. Therefore some participants accidentally pressed the spacebar for too long while typing a space between words while in the middle of an entry. To prevent these mistakes from interfering with my analysis, I removed 15 entries in which a participant was missing three or more words at the end of a phrase. Also, due to a logging bug, I had to drop another 38 entries. At most, this bug occurred three times to any participant in any condition's set of 20 phrases. My analysis was on the remaining 1,867 entries from the original set of 1,920.

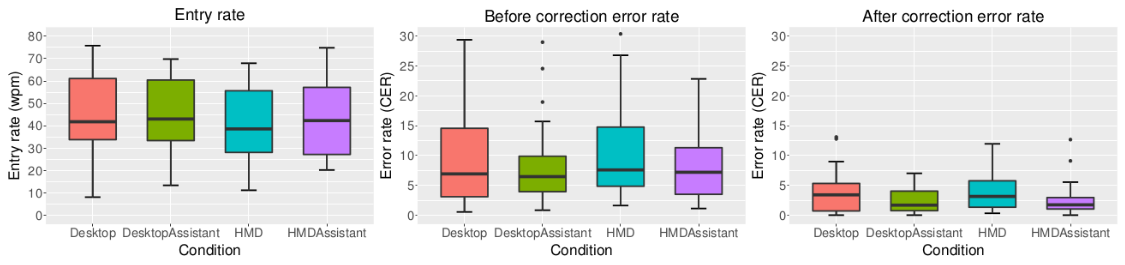


Figure 2.4. Participants' average entry rate (left), error rate before automatic correction (center), and error rate after automatic correction (right).

Table 2.2. Participants' average performance in each condition in the study. Results are formatted as: mean ± sd [min, max].

Condition	Entry rate (wpm)	Before CER (%)	After CER (%)
DESKTOP	44.7 ± 18.6 [8.1, 75.8]	11.2 ± 11.0 [0.5, 40.9]	3.9 ± 3.7 [0.0, 13.1]
DESKTOPASSISTANT	44.7 ± 16.3 [13.4, 69.8]	8.3 ± 7.3 [0.8, 29.0]	2.6 ± 2.2 [0.0, 7.0]
HMD	41.2 ± 17.5 [11.2, 67.9]	11.8 ± 11.4 [1.6, 49.6]	4.0 ± 3.2 [0.3, 11.9]
HMDASSISTANT	43.7 ± 17.0 [20.2, 74.8]	8.4 ± 6.2 [1.1, 22.8]	2.6 ± 2.9 [0.0, 12.7]

Table 2.3. Details of two-way repeated measures ANOVA. Significant differences highlighted in bold.

Entry rate (wpm)	Feedback	$F_{1,23} = 2.10$	$p = .16$	$\eta^2 = 1.5 \times 10^{-3}$
	Display	$F_{1,23} = 7.79$	$p < .05$	$\eta^2 = 4.4 \times 10^{-3}$
	Feedback × Display	$F_{1,23} = 3.61$	$p = .07$	$\eta^2 = 1.4 \times 10^{-3}$
Before correction error rate (CER %)	**Feedback**	$F_{1,23} = 7.21$	$p < .05$	$\eta^2 = 2.9 \times 10^{-2}$
	Display	$F_{1,23} = 0.22$	$p = .64$	$\eta^2 = 4.2 \times 10^{-4}$
	Feedback × Display	$F_{1,23} = 0.15$	$p = .70$	$\eta^2 = 2.4 \times 10^{-4}$
After correction error rate (CER %)	**Feedback**	$F_{1,23} = 6.78$	$p < .05$	$\eta^2 = 4.9 \times 10^{-2}$
	Display	$F_{1,23} = 0.02$	$p = .90$	$\eta^2 = 6.8 \times 10^{-5}$
	Feedback × Display	$F_{1,23} = 0.02$	$p = .89$	$\eta^2 = 1.0 \times 10^{-4}$

I tested for significance using a two-way repeated measures ANOVA. The two independent variables were whether the keyboard assistant was shown or not (denoted Feedback), and whether visual feedback was provided on an HMD or on a desktop monitor (denoted Display). Details of my statistical analysis appear in Table 2.3.

Entry Rate

Entry rate is reported in words per minute, defining a word as five characters including spaces. Entries were timed from the first key press until the recognition result was displayed to the user. This includes the time required for a user to perform the long keypress, networking delays, and recognition delays. In this study, network and recognition delays averaged 0.35 seconds per phrase (standard deviation 0.71).

Entry rates were similar across all conditions. Using the monitor and keyboard cover, participants achieved an entry rate of 44.7 wpm both with and without the virtual keyboard assistant. While wearing the HMD, participants typed at 41.2 wpm without the virtual keyboard assistant and 43.7 wpm with the assistant (Figure 2.4 and Table 2.2). Participants’ slower entry rates using an HMD display were significant (Table 2.3). The similar or faster entry rates in the presence of the virtual keyboard assistant were not significant.

Error Rate

I report typing and recognition accuracy using character error rate (CER), using the same definition as previous. I report the before and after correction error rate. The before correction error rate used the literal text the participant typed before any autocorrection. The after correction error rate used the recognition result of the decoder.

Before correction error rates were lower in conditions with the virtual keyboard: 8.3% with the covered keyboard and 8.4% with the HMD versus 11.2% with the covered keyboard and 11.8% with the HMD (Figure 2.4 and Table 2.2). Participants' lower before correction error rates using the virtual keyboard assistant were significant (Table 2.3). There was not a significant difference in before correction error rates between the HMD and desktop display.

Conditions with the virtual keyboard also had a lower error rate after decoding: 2.6% in both virutal keyboard conditions versus 3.9% with the covered keyboard and 4.0% with the HMD without the virtual keyboard. Participants' lower after correction error rates using the virtual keyboard assistant were significant (Table 2.3). There was

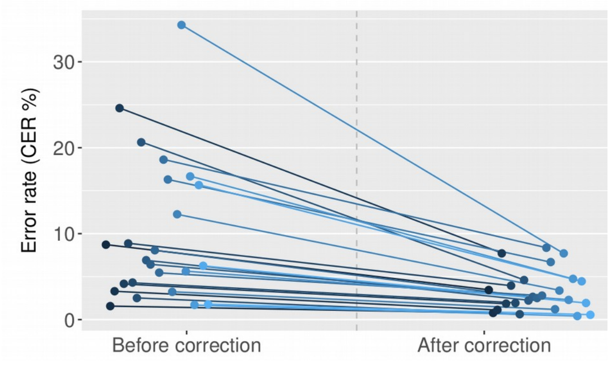


Figure 2.5. Participants' average error rate across all conditions before and after correction.

not a significant difference in after correction error rate between the HMD and desktop display.

Before correction error rates were quite variable by participant as shown in Figure 2.5. The decoder reduced error rates for all participants, with some participants experiencing large gains in accuracy. After correction, all but four participants achieved an error rate of less than 5%.

Recall that I removed 15 entries from my analysis due to participants erroneously ending their sentence three or more words early. I also conducted statistical analysis on the data without these entries removed. All the statistical conclusions were the same. The main difference in results was a somewhat elevated before correction error rate: 11.5% with the covered keyboard and no virtual keyboard assistant, 8.8% with the covered keyboard and the assistant, 12.8% with the HMD and no assistant, and 8.5% with the HMD and the assistant enabled. The after correction error rate was also somewhat elevated: 4.3% with the covered keyboard, 3.1% with the covered keyboard with the assistant enabled, 5.2% with the HMD, and 2.7% with the HMD and the assistant enabled.

These results show that the live feedback offered by the virtual keyboard assistant significantly reduced participants’ error rates. Even though my feedback was of already committed keystrokes, users were still able to leverage this feedback to improve performance. I conjecture the feedback may have allowed users to correct mistakes such as offset hand position, making them more accurate from that point on. It may also have provided a visual reminder of the QWERTY layout. Because most participants self-reported as touch typists, these results are only applicable to proficient typists.

The autocorrection algorithm clearly improved users' accuracy across all conditions, even in the face of highly inaccurate input. Typing while wearing an HMD was slower, but not substantially more error prone than typing in a more normal desktop situation with a visually occluded keyboard. I conjecture that HMD entry rates were slower due to unfamiliarity with the device, but since the visual information was largely the same across displays, error rates were similar. It remains an open question why the HMD seemingly affected speed but not accuracy.

I only tested entry of the letters A–Z and the apostrophe. I think my approach of a virtual keyboard assistant might be especially useful in the case of entry of less common characters such as symbols, numbers, or chorded keystrokes, though such text would also be more difficult to recognize accurately.

2.5. Summary

One of the challenges regarding the usability of HMDs is the difficulty of using standard physical input devices while wearing them. In order to investigate mitigation strategies for this, I devised a system to aid users in typing on mechanical keyboards while wearing HMDs. The initial system utilized VelociTap, a touchscreen decoder which I modified for use with physical keyboards. I used data gathered from a pilot study utilizing this system to further improve my system with additional modifications to VelociTap and a virtual keyboard assistant. I then performed a larger follow-up study to thoroughly evaluate my text entry system.

My findings showed that a virtual keyboard providing live feedback coupled with an autocorrection algorithm substantially improved users' typing performance while wearing an HMD. A notable advantage of my approach is that it does not require any tracking devices or external cameras, making it readily usable without additional hardware. This could be a significant factor for some users who are on a budget or do not have the space to install additional devices. A system that does not "intrude" on the virtual environment by superimposing real-world imagery over the virtual imagery might also be less distracting for HMD users. These findings demonstrate that software-only solutions for improving users' ability to interact with physical devices while wearing an HMD show considerable promise and merit further investigation as research in this field intensifies.

Chapter 3: VR Windowing System: A Fish Out of Water

3.1. Introduction

Despite low-cost virtual reality interface technology having entered the consumer market, to date there has been relatively little investigation into the “mundane” applications of virtual reality, such as its possible uses on a home or office desktop environment. This creates the possibility of new technology flooding the market without any clear uses for it outside of the obvious entertainment applications. These potential uses are worth investigating because virtual reality interface technology could provide new and original approaches to the most fundamental tasks users perform on their computers, such as application management.

This work investigated the development of software optimized for HMDs for use with everyday home and office tasks. This is not a merely theoretical area of investigation. Users have already expressed interest in utilizing head-mounted display (HMD) technology to manage their desktops and applications. Here are some user comments in response to an experimental HMD window manager [39] posted on Reddit [59]:

- “Wow. This is something I'd hoped to see the Rift used for. I'm looking forward to replacing my desk full of monitors with a single headset.”

- “Imagine having an office workspace with no physical displays at all, just a Rift and you're done… you're surrounded by a relaxing environment like a some waterfalls [sic] in a jungle amidst which your terminal windows float around, awaiting your input”
- “imagine that...instead of paying a grip for 3 30" monitors or whatever you could sit there and design the ultimate virtual workstation or entertainment center... so cool.”

Other examples of burgeoning HMD usage can be found. For instance, Bloomberg has created a prototype virtual terminal for use with the Rift [45]. Or consider the fact that most initial purchases of consumer HMDs for home use will be for entertainment purposes; i.e. virtual reality video games. Switching between using the HMD for games and a standard monitor for other desktop applications could be burdensome. An HMD-optimized window manager would allow users to switch between their entertainment and utility applications without needing to also switch display hardware, or suffer through the ordeal of trying to use a standard window manager in an HMD.

3.2. Background and Related Work

The task of managing multiple applications on a traditional desktop is accomplished using a special type of application called a window manager. Window managers come in multiple styles. The most common types of traditional window managers are *stacking* and *tiling*. Stacking window managers allow users to place and size windows at will, including placing windows on top of each other so that their areas overlap. By contrast, tiling window managers do not allow overlap and aid the user in organizing windows into tiles on the screen.

There are also *compositing* window managers, also called compositors. Compositors store applications' visual contents in a graphical buffer so that they can be subjected to sophisticated graphical manipulations (e.g., using OpenGL) in order to achieve striking visual effects and enable three-dimensional interface elements. Rather than a category in itself, compositing is more of a property of window managers. A window manager could be both stacking and compositing, for example.

On current Linux systems, window managers must utilize the X protocol and are typically coded either against the X library itself or an intermediate API such as XCB. Because of the X library's age, it was not designed with 3D graphical capabilities in mind. This capability has been added in an ad-hoc manner using various extensions, which makes writing compositing window managers for X a convoluted process. Alternatives exist, such as the Wayland protocol, but at the time of development, these were in experimental stages and thus did not serve as a stable development platform [58].

Users are always striving for more efficient ways to simultaneously manage large numbers of applications. Many users prefer large displays, or multiple displays, for this purpose. Research has shown that users are often more productive when given larger display areas to work with, yet there are also disadvantages compared to a traditional monitor; for example, large amounts of time spent repositioning and resizing windows, accidentally closing windows, and difficulty navigating a large display area with a mouse and keyboard [9, 10, 22, 41, 46]. These findings simultaneously demonstrate the utility of large display areas and the problems that arise from using traditional interface technologies to present large displays to the user.

HMDs, with their ability to display a virtual environment that entirely surrounds the user, present an opportunity to move beyond the limitations of flat monitors. Some companies and researchers have already recognized the potential in this area. An application called Virtual Desktop purports to serve this purpose [52], but in actuality it merely projects a verbatim rendering of the user’s desktop onto a textured quad and places it within a decorated virtual space. Although the effect can be aesthetically pleasing, this simple approach does nothing to utilize the potential of a true VR window manager.

Purpose-made VR window managers remain elusive. A cursory Internet search reveals a few such projects in development, but as of this writing, there do not appear to be any finished, usable products on offer. One of the more mature offerings, called Motorcar, was developed by Reiling [39]. Because Motorcar bears the closest resemblance to the work described in this chapter, it warrants a more detailed examination.

Motorcar is fundamentally a compositing window manager developed using the Wayland protocol. The primary objectives of Motorcar are to enable the free arrangement of application windows in three-dimensional space, to provide transparent support for both 2D and 3D applications so that applications displayed in Motorcar can be developed in a hardware-agnostic manner, and to be designed for use with HMDs as the primary display device, rather than flat monitors. However, while Motorcar serves as an effective tech demo and proof-of-concept, its lack of restrictions on placement and positioning of windows is likely a drawback rather than an asset. Some of the user comments in response to a video of Motorcar in use echo this sentiment, with such comments as, “[T]hat looks excruciating,” and “Neat to look at but impossible to use for any real tasks” [59]. While Motorcar is a step in the right direction, an HMD-optimized windowing system needs a greater focus on usability and user studies to evaluate the effectiveness of different interface approaches.

Another similar project is the Personal Cockpit developed by Ens et al [14, 15]. The Personal Cockpit is a virtual window manager designed for use with see-through HMDs; and unlike the majority of window manager research explored, this work includes a user study evaluating the system's effectiveness, with favorable results. However, it differs from my work in that it relies on the user being able to manipulate the virtual windows by direct touch, which necessitates a system to track the user's hands. This places the system out of reach for any users who do not have ready access to such tracking systems.

Other window management systems with some similar features, but different goals, have been developed. Robertson et al [42] developed a 3D window manager for Microsoft Windows called the Task Gallery that utilizes users' spatial memory and bears some similarities to a virtual environment desktop. Topol [48] presents a similar system for the X11 windowing system that organizes application windows in a virtual environment-like space. A similar but more sophisticated system called 3DWM (Three-Dimensional Workspace Manager) was developed by Elmqvist [13]. However, none of these systems are appropriate for desktop HMD use. Both the Task Gallery and Topol's system were designed for use with traditional monitors and do not support sensor input as a control mechanism. 3DWM is essentially a hardware abstraction layer and toolkit to facilitate development and research of 3D user interfaces rather than an effective interface in itself.

There has also been similar work in the field of augmented reality. Feiner et al [16] developed an augmented reality window manager for use with see-through HMDs, but their intent was to create a virtual heads-up system rather than an effective desktop window manager. DiVerdi [11, 12] developed an augmented reality window manager called ARWin with a focus on displaying applications as though they were an extension of the user's physical desk.

Finally, another subgenre of VR research that bears some similarity to this work is virtual/augmented reality menu systems. Although menu systems in isolation are not as robust as a full window manager, most window managers include some sort of menu system as part of their functionality. Common types of menu systems include floating menus, which are similar to standard 2D menus familiar to most users except that they float in 3D space; pen and tablet menus where interface components are placed on a tablet or similar surface and interacted with via a stylus or similar device; and gesture-based menus which respond to user movements, often aided by a tracked device such as a glove [4]. Gesture-based menus have received a particularly large share of attention in recent years. A few examples include the Pinch Glove by Bowman and Wingrave [4]; the WeARHand system, an AR gesture-based selection system that tracks users' bare hands with HMD-mounted RGBD cameras [21]; and the vDesign system for use with CAVEs [37]. More exotic VR menu systems have also been implemented and tested, such as pie menus activated by a 6DOF pointing device [20].

Overall, while HMDs show obvious potential for desktop use, research into this area it still preliminary in nature, with few studies objectively evaluating effective techniques. Thus, this was an area ripe for investigation.

3.3. Development

To investigate how the unique features of HMDs can be leveraged in a typical desktop environment, I developed a prototype HMD-optimized window manager called Fish Out of Water, or FOW for short. This section will discuss highlights from the development process, as FOW went through numerous changes during implementation.

Both the X protocol and Wayland were investigated as possible frameworks on which to build a virtual reality window manager. However, X's age makes 3D graphical rendering extremely convoluted and Wayland is still in an experimental stage. Development using either platform is therefore a very time-consuming process. Because the objective of this research was not to create a release-quality product that could be deployed “out of the box” on existing systems, but rather to explore and evaluate the effectiveness of HMD interface techniques for common computing tasks, the time-intensive nature of X and Wayland development resulted in both being rejected as development platforms for FOW.

Instead, a prototype FOW system was developed using Processing, which is essentially a subset of Java with a convenient built-in API for interacting with OpenGL as the graphical back-end [38]. The first version of FOW used AutoHotkey scripts to interact with the operating system and control other applications. The prototype utilized stereoscopy to achieve true 3D using an HMD and also applied lens distortion correction with a GLSL fragment shader. The prototype also interfaced with a DK2 HMD using a relay program developed with the Oculus SDK2 that forwarded the HMD's orientation data to the window manager via UDP packets. The basic features of application manipulation and minimizing applications into thumbnail windows were implemented by manipulating applications on a separate screen utilizing the AutoHotkey scripts, grabbing the window graphics with the Java Robot class, and then rendering them in the virtual desktop environment as textured quads (a similar approach to Virtual Desktop, but on a per-window basis instead of grabbing the entire desktop).

This approach cut development time to a small fraction of what would be required with X or Wayland, with the drawbacks that FOW could only be deployed on Microsoft Windows systems and was quite demanding on hardware resources. This comprised the minimal set of features needed to constitute a functioning window manager, but at this stage, many minor details and an attractive graphical representation had yet to be developed.

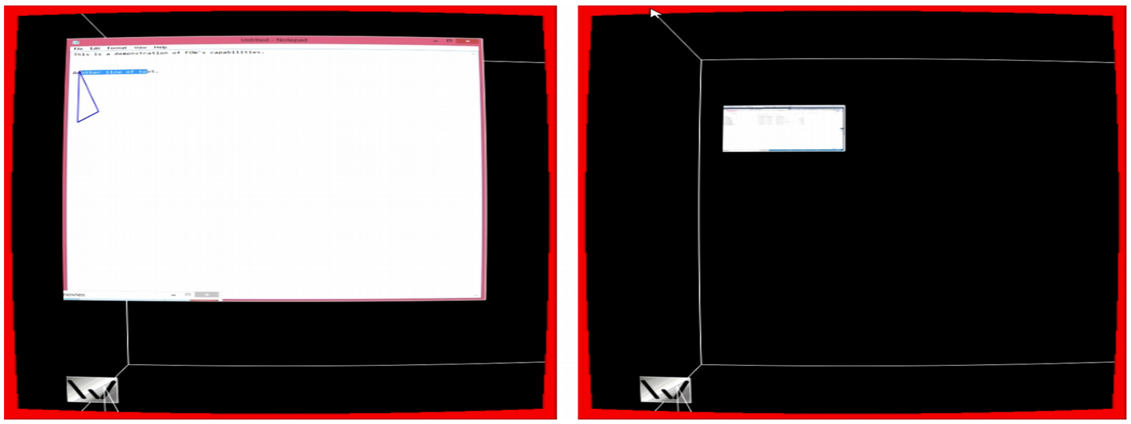


Figure 3.1. Screen captures of an early prototype of FOW. The left screen capture shows Notepad being stereoscopically rendered on a textured quad as a user interacts with it live. The right screen capture shows the Notepad window minimized to thumbnail size.

Eventually, I decided to abandon many aspects of the approach taken by the first version of FOW. Although it was fully functional, this implementation suffered from two crucial drawbacks. First, as mentioned, the process of grabbing parts of the screen and rendering them as textured quads was computationally intensive, resulting in poor performance of about 20 frames per second (fps). This performance was thought to be unacceptable for typical desktop use. Secondly, although interacting with real applications made FOW usable as a true window manager, this minimized the amount of control that I had over the applications themselves, which would be an impediment to running an effective, tightly controlled study. Since the primary purpose of FOW was to serve as a research vehicle rather than a production-quality application, this did not seem like the most optimal approach.

Subsequently, a second version of FOW was developed which also used Processing and GLSL shaders for rendering. However, the AutoHotkey scripts and use of real applications were scrapped. Instead, “fake” applications were developed in the form of textured quads that the user could interact with by dragging them around, scrolling over them with sliders, and entering text on designated quads through keyboard input. This gave me full control over the artificial desktop environment for research purposes. The virtual keyboard assistant described in Chapter 2 was also reimplemented in FOW, with the option of hiding (and reshowing) it with the tab key. The VelociTap decoder was not included, however, because a prohibitively large amount of work would have been required to alter the decoder to effectively correct arbitrarily long blocks of text that could include non-alphabetical characters, particularly numbers. The second version of FOW also tracked the user’s head rotation using the Vicon tracking system in the Immersive Visualization Studio with Kalman filtering enabled rather than using the DK2’s built-in sensor.

With this new framework in place, remaining features were implemented. In order to make the manager more usable, FOW does not permit users to place windows anywhere in 3D space. Instead, a conceptual cylinder exists around the user, and windows can be placed anywhere along the surface of this cylinder. In this way, windows will always be placed such that it is possible to view them straight-on simply by rotating one’s head, while the user still has the freedom to place windows anywhere in a 360-degree circle around himself, and can also move the windows vertically to any location that he finds convenient. Lastly, for aesthetic purposes, a background resembling an overhead view of distant terrain with cloud cover was added. At first, this background was made to rotate to create the illusion of being in orbit over a planet, but the rotation was scrapped when initial tests showed that it caused motion sickness in some users (see Figure 3.4).

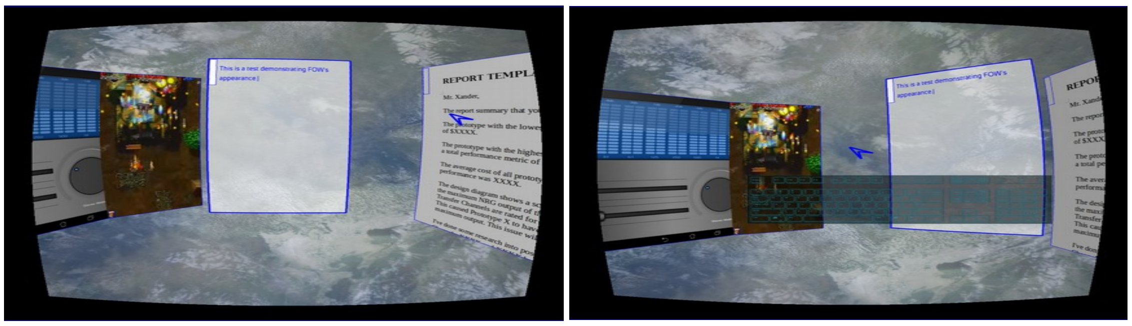


Figure 3.2. Screen captures of the second version of FOW with the virtual keyboard hidden (left) and shown (right). The virtual keyboard is positioned for optimal visibility in a DK2 HMD.

3.4. Survey

I collected feedback for FOW with a user experience survey. For this survey, participants used FOW to perform a simulated everyday computing task (composing an email), then answered a questionnaire to provide feedback about their experience. The results of this survey confirmed that participants found the extra space afforded by the 360-degree virtual environment convenient and useful, while also exposing areas where FOW could use improvement.

Design The study was designed to simulate the everyday computing task of composing an email while also forcing participants to switch between many applications, since the hypothesis is that the extra space afforded by a virtual environment eases the burden of switching between applications.

Within the virtual space, multiple windows were displayed. One of these windows was a simple text editor, initially blank, in which the participant could enter text. Another window contained the text for an email to a fictional supervisor, summarizing fictional research results. Some of the text of this email was replaced with capital Xs; for example, “The prototype with the lowest net production cost was X, with a cost of $XXXX.” The participant was to copy this email into the text window. Wherever the email contained Xs, the participant needed to find the relevant information, which was contained in other windows, and enter that information in place of the Xs. (See Figures 3.5 and 3.6.) This design ensured that participants would have to switch between windows many times in order to complete the task.

Procedure Participants were informed of their task, as described in the previous subsection. They were also instructed in FOW’s functions, including text entry, moving windows, using the slider, and so on. Participants were told to use whatever strategy they thought was appropriate in regard to using FOW and arranging windows in order to complete the task in an efficient and convenient manner. In addition to written and verbal instructions, the experimenter also gave a live demonstration of how to use FOW before turning over control to participants.

While participants completed the task, FOW automatically logged information of interest, including total time taken, total head rotation, number of windows moved, and text entered. After completing the task, participants were given a questionnaire so they could provide feedback about their experience. This questionnaire asked participants several usage questions using a Likert scale from 1 (strongly disagree) to 5 (strongly agree), asked them to report simulator sickness symptoms, and collected feedback with six open-ended questions.

Results and Discussion 13 participants took part in the survey (6 female, ages 18-30). The results from the Likert scale questions, automatic data logging, and self-reported simulator sickness are given in Tables 3.1 through 3.3. The complete text of participants' responses to the open-ended questions can be found in Appendix B.

Despite being unable to see the real world while using FOW, on average, users expressed only moderate difficulty keeping track of the physical mouse and keyboard (mean 2.85), although some users did mention that this issue was a mild hindrance in the open-ended questions. This finding suggests that being unable to see physical peripherals while wearing an HMD is not necessarily a show-stopper for usability, even without a sophisticated camera system and software to superimpose these details over the virtual environment.

Surprisingly, on average, users were neutral about whether they had enough space for all the application windows (mean 3, and a 3.23 on the related statement, "Having 360 degrees of space to position the windows was convenient"), although a median and mode of 4 indicates that the average was pulled down by a handful of low ratings. Relatedly, users did not report great difficulty in finding the windows they needed (mean 2) despite the large number of windows required for the task.

Table 3.1. Results from FOW survey Likert scale questions.

	Mean	Median	Mode
I had difficulty keeping track of the physical mouse and keyboard.	2.85	3	2
I had enough space for all the application windows.	3	4	4
I had a hard time finding the windows I needed.	2	2	1
Having 360 degrees of space to position the windows was convenient.	3.23	3	4
I frequently lost track of the cursor while working.	3	3	2
The virtual keyboard helped improve my typing performance.	3.46	4	5
I found it hard to switch between windows.	2.54	2	2
I made use of vertical space (i.e., placing windows above/below each other).	3.23	4	1

Table 3.2. Results from data logged by FOW.

	Mean	Min	Max
Time to complete task (seconds)	1778	984	3563
Total horizontal head rotation (radians)	277	95	480
Text deletions	235	61	929
Virtual keyboard on (%)	77.72	5	100

Table 3.3. Users' self-reported simulator sickness symptoms. 0 = None, 1 = Slight, 2 = Moderate, 3 = Severe.

	Mean	Min	Max
General discomfort	0.69	0	2
Fatigue	0.54	0	2
Headache	0.54	0	3
Eyestrain	1.00	0	2
Difficulty focusing	0.77	0	2
Increased salivation	0.15	0	1
Sweating	0.46	0	2
Nausea	0.38	0	2
Difficulty concentrating	0.53	0	2
Blurred vision	1.23	0	2
Dizzy (eyes open)	0.46	0	2
Dizzy (eyes closed)	0.23	0	1
Vertigo	0.30	0	2

In addition, several users explicitly mentioned in both the open-ended questions and verbal discussion that the large amount of space afforded by the 360-degree virtual environment was more convenient than a traditional desktop for managing many applications at once. This suggests that, as expected, the large space afforded by FOW's design is useful for handling many applications, but a significant minority of users may be unable to leverage this capability to good effect, at least without practice.

In general, users were enthusiastic about the virtual keyboard's ability to improve their typing performance (mean 3.46), with the most common response being 5, Strongly Agree. This is also reflected in the logged data, which shows that on average users kept the virtual keyboard on 77.72% of the time, with 9 out of 13 users keeping the keyboard on over 98% of the time, despite being encouraged to disable it if they found it unnecessary or distracting.

Users took an average of about half an hour (29.63 minutes) to complete the task, with the fastest user taking 16.4 minutes and the slowest an hour (59.38 minutes). Users rotated their heads an average of 277 radians, or 44 full rotations. An average of 235 characters were deleted (the finished email was approximately 1,055 characters). Users' text entry accuracy rates were encouraging. Precise accuracy metrics were not calculated because of multiple possible interpretations for how to correctly format the email. However, running a diff command on each user's final output against the correct output indicates that 10 out of 13 users had accuracy rates of about 99% (in many cases, clearly achieved thanks to significant corrections and editing, as evidenced by the high character deletion rate). One user had output with long sequences of “garbage characters” which were not human-readable, presumably due to a lack of touch-typing proficiency. Other errors which caused accuracy rates significantly below 100% were due to simple editing mistakes such as missing a sentence. For future work, it would be interesting to have users perform the same task on a normal desktop and compare these metrics.

The average reported severity of almost all simulator sickness symptoms was less than 1 (between None and Slight). The only exceptions were eyestrain (mean 1.00) and blurred vision (mean 1.23). This result is expected due to imperfect chromatic aberration correction, which causes some graphics (especially small text) to appear blurred at the edges of the display; and because the DK2’s resolution is not optimally suited for reading small text for extended periods of time. Aside from these issues, most users did not experience an abnormal amount of simulator sickness from using FOW. There was only one occurrence of a Severe (3) symptom: one user reported a severe headache from using the system.

Users provided useful feedback and observations through the open-ended questions. Many users commented on the convenience of having 360 degrees of space for placing windows. The simplicity of being able to view many windows simply by turning one’s head, rather than having to deal with minimizing and maximizing or alt-tabbing through many applications, elicited a significant amount of positive feedback. However, two users suggested that being able to recenter the view would be more convenient than moving multiple windows to their desired locations. This would be an excellent feature to add to a future version of FOW.

Users also had a great deal of praise for the virtual keyboard's usefulness in improving their typing performance, which is backed up by the virtual keyboard's usage logs. However, some users expressed a desire to be able to move the virtual keyboard from its default position.

Physical limitations arising from the DK2 itself drew some criticism. My shader for correcting the DK2's chromatic aberration was not perfect, resulting in some blurring around the edges. This blurring resulting from the DK2's relatively low-quality lenses drew more criticism than any other single aspect of the system, and this is borne out by the nearly universal reporting of eye strain and blurred vision in the simulator sickness questions. The resolution of the DK2 was also a contributing factor and was explicitly mentioned by at least one user. This suggests that for performing tasks which involve reading relatively small text, having high-quality lenses and extremely high resolution would be desirable in order to avoid causing significant user discomfort. Unfortunately, HMDs manufactured to the necessary level of quality remain prohibitively expensive for consumer use, but this is likely to change with time.

Aside from the limitations of the HMD, elements that drew the most criticism were behavioral limitations arising from FOW's status as experimental research software. For example, the text editor was primitive and supported only the most basic text editing functionality, which some users found inconvenient. Some users also expressed a desire to be able to resize windows, functionality which was absent in FOW. Many users found the sliders for navigating windows difficult to use. In response to this common criticism, any future iterations of FOW or similar software should focus on making the sliders larger and more responsive. In general, implementing various common window manager features which were absent in FOW, such as window resizing and common shortcut keys, would improve its usability.

In addition, some users disliked the fact that overlapping windows intersected each other on the Z axis (due to all windows being displayed on the surface of an invisible cylinder), and suggested that being able to order windows' layers on the Z axis would improve usability.

The complete text of users' responses to the open-ended questions can be found in Appendix B.

3.5. Summary

In response to the gap in virtual reality research for everyday computing tasks, I developed Fish Out of Water, or FOW, a prototype window manager designed for use with HMDs. FOW works by placing windows on the inner surface of an invisible cylinder and allows users to move windows anywhere on this surface, providing 360 degrees of usable space. I performed a user experience survey to evaluate FOW’s usability. Users reported that the large amount of space was convenient, confirming my expectations; and users also responded positively to the virtual keyboard’s ability to improve their typing performance. Most user criticisms revolved around the DK2’s chromatic aberration (and my imperfect correction of it) and resolution, as well as some features missing from FOW due to its prototype status. These findings suggest that there is great potential for window managers optimized for use with HMDs, but that display technology (particularly resolution and lens quality) might need further improvements in order to make optimal use of this potential.

Chapter 4: It’s Not Where You Think It Is: Illusory Distances

4.1. Introduction

Many studies have shown that users consistently underestimate distances in virtual environments while wearing HMDs [62, 63, 26, 47, 43, 1, 61, 60, 30]. The reasons for this are still not fully understood. Perceiving the virtual environment as closely as possible to the real-world equivalent is important in many virtual reality applications. This is most apparent in safety-critical or modeling applications, but even everyday computing tasks could be adversely affected by inaccurate distance judgments, since a sophisticated window manager or similar productivity application might make heavy use of the depth dimension, which could be undermined if users are not perceiving distances as the developer intends. Accurate distance perception is thus another important factor in the usability of virtual environments. In this chapter, I discuss research that I have conducted investigating distance perception in virtual environments and potential techniques for improving users’ distance perception.

4.2. Background and Related Work

One important consideration when researching virtual distance perception is that it is not possible to measure users’ perception of distances directly. Techniques must be devised to indirectly measure the accuracy of users’ distance judgments. The most obvious method is to ask users how far away they think a given object is, a technique called verbal report.

Another technique, called blind walking, shows the user an object, then removes the user's vision in some way (such as blacking out the graphics) and has the user walk to where the user thinks the previously perceived object is. Other techniques include indirect blind walking, in which the user walks to an intermediate position which is not colinear with the user's starting position and the target before turning to walk to the target; and throwing a bean bag to the perceived target location after having vision removed.

With all of these techniques, users have been shown to significantly underestimate distances in virtual environments [62, 63, 26, 47, 43, 61, 60, 19]. This is not due to an inherent inaccuracy in distance judgments, because other studies have shown that humans accurately judge distances as measured by blind walking in the real world [40, 33, 63].

Many possible reasons have been proposed for users' compressed distance judgments, such as reduced peripheral vision when wearing HMDs, quality of the graphics in the virtual environment, or the weight and momentum of HMDs. For a long time, none of these factors were shown to significantly explain compressed distance judgments [47, 60, 26, 8, 65, 25]. Recent work has shown that a bright frame can improve distance perception when the field of view is small [32], but the following research was conducted before this solution was well-examined.

4.3. The Effects of Minification on Distance Perception in HMDs

4.3.1. Background

This work explored the use of minification to affect users' distance judgments in HMDs. Minification can be thought of as the opposite of magnification. When the graphics are minified, the rendered field of view is expanded while the dimensions of the physical display remain the same. This has the effects of increasing how much of the scene the user can see and making everything in the scene look smaller. See Figure 4.1.

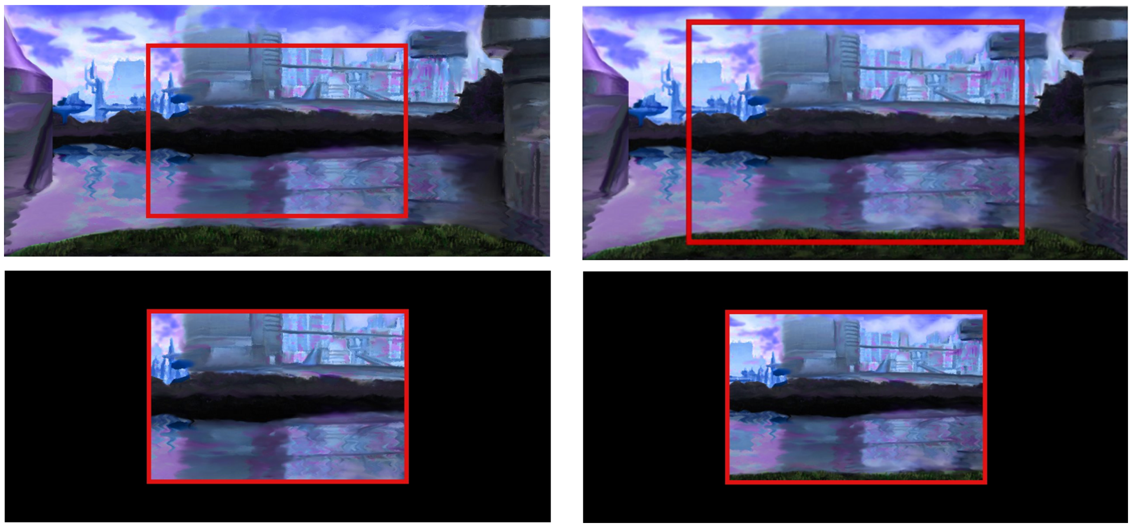


Figure 4.1. The red frame in the upper two images represents different fields of view within the same scene. The upper-right image represents a minified field of view. The bottom two images represent a physical display of the same size, and how the scene would be rendered using the field of view shown above that image.

One might note that minifying a scene has a similar effect as translating backwards, placing the user farther away from the rendered imagery. However, the effect of these two transformations on the visual scene is not identical, so these transformations are not interchangeable. If the user’s position is translated backwards, this translation could move the user through a wall, place the user’s center of rotation outside the user’s body, and cause various other undesirable side effects. As a result, backwards translation is not an acceptable transformation to apply in virtual environments for most use cases.

Figure 4.2 further illustrates the differences between minification and backwards translation. The left image (A) shows the user’s default position. Here, the user’s view is completely blocked by the wall. In the center image (B), the user’s rendering position has been translated backwards compared to the user’s true position. In the right image (C), the user’s field of view has been minified. The transformations in B and C have been applied such that the same amount of the back wall (the top of the grid) is visible in each case, yet the amount of the scene that is visible is not the same. The green shading in image B shows imagery that is visible to the user in B but not C. In addition, much more of the user’s vision of the back wall would be occluded by the intervening wall in image C than in B.

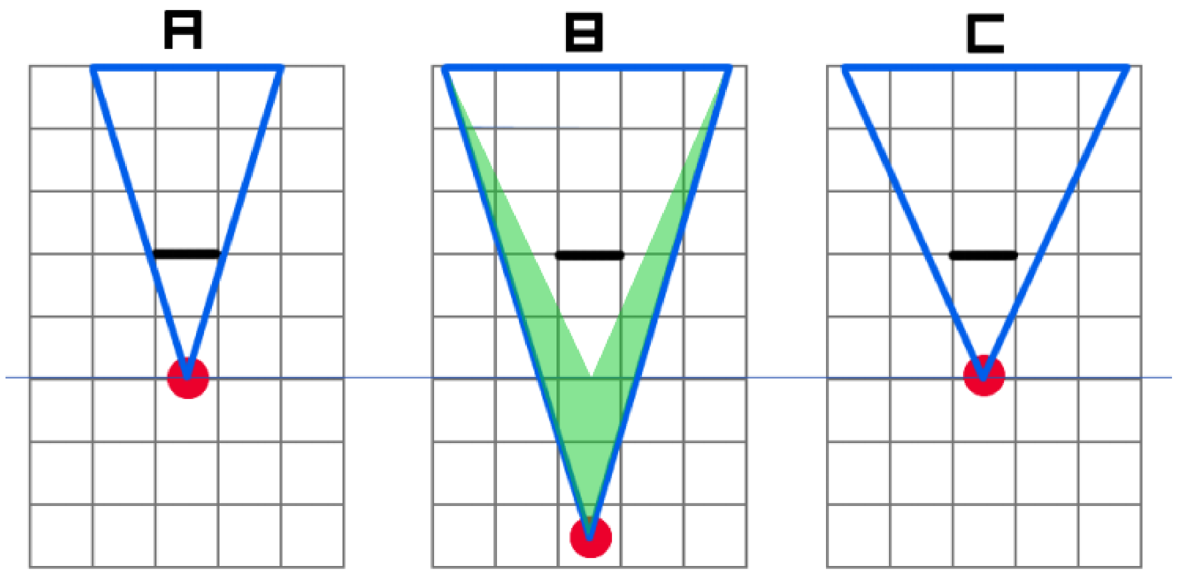


Figure 4.2. An illustration of the effects of minification vs. backwards translation. This image represents an overhead view of a virtual environment. The red dot represents a user's position, the blue triangle shows the user's field of view, and the solid black line represents a wall in front of the user.

4.3.2. Study

I conducted a between-subjects study to investigate the effects of minification on users' distance judgments in HMDs [67], based on previous work showing that minification could reduce or eliminate distance compression [28, 29]. However, this previous work was limited to blind walking in narrow hallways. To more fully understand the effects of minification, a new study was conceived to evaluate minification in a more open environment and to evaluate verbal reports in addition to blind walking.

For this study, participants were shown a virtual room modeled after the virtual reality lab in the Rekhi building at Michigan Technological University. The hardware used was an NVIS nVisor ST HMD, with four WorldViz PPT-H cameras to track the position of the

HMD and an InertiaCube2 inertial measurement unit to track the orientation. I measured a default field of view in the HMD of 47.4 × 39.85 degrees and applied minification by expanding the field of view to 54.75 × 64.18 degrees. The size of the physical lab used for the study was 10 × 8 meters.

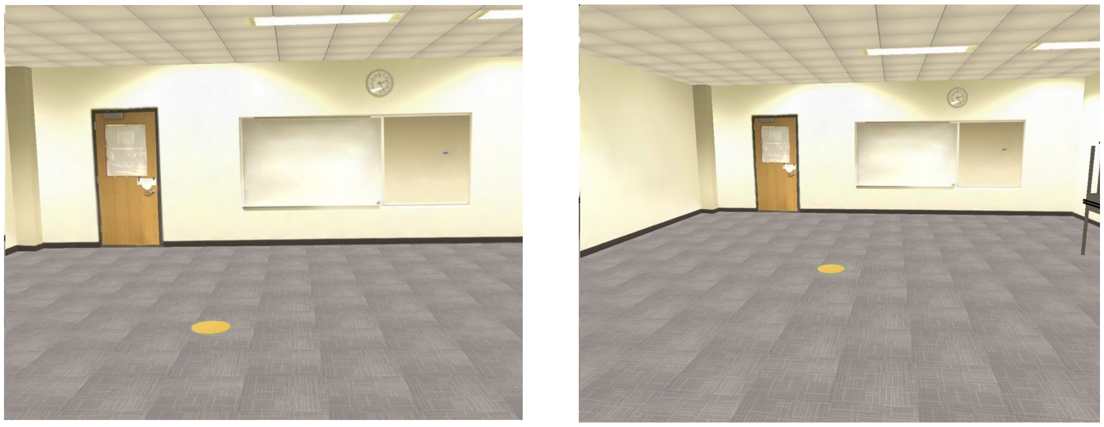


Figure 4.3. View of the virtual environment in the normal (left) and minified (right) conditions.

The study consisted of two separate experiments. The first experiment measured participants' distance judgments by blind walking, and the second by verbal reports. In both experiments, participants of age 18 to 35 were given credit through the psychology participant pool or paid $10 for their participation. After participants provided informed consent, they were screened to ensure that they had at least 20/30 visual acuity and were not stereoblind by identifying a 3D shape on a random-dot stereogram. Participants wore contacts or glasses if they were needed for corrected-to-normal vision.

This screening occurred outside the lab where the experiments were conducted. After the experimenter explained the procedure, each participant wore a blindfold and walked around outside the laboratory for several minutes, guided by the experimenter, in order to become accustomed to walking without being able to see the real world. After this process, the participant was brought into the laboratory with the blindfold on, so that the participant did not have any prior knowledge about the physical environment before the experiment. Noise canceling headphones with white noise blocked acoustic cues. Participants wore the HMD at all times, viewing the virtual room modeled after the lab where the study took place. The graphics were rendered with either a normal or minified field of view depending on the condition the participant was assigned to.

For the first experiment, participants were shown colored targets rendered on the floor of the virtual scene. Then the graphics in the HMD were blacked out and the participants walked to where they believed the target was located and stopped. Then they were guided back to the starting position, the next target was loaded, and the graphical view was restored. This process repeated until the participant had completed all the trials. The first three trials consisted of a random ordering of targets at 2.5, 3.5 and 4.5 meters. The next 15 trials consisted of the distances 2, 3, 4 and 5 meters displayed three times each and the distances 2.5, 3.5 and 4.5 meters displayed one time each. The order of these trials and the size, color, and shape of the targets were randomized. After the participant walked to the target and stopped, they kept their eyes closed and were guided back to the starting position in the real world.

The participant always started from the same real world location and the entire virtual environment was translated randomly (without translating the target) to make it appear that the virtual starting location had changed. The HMD screens were blank whenever the participants' eyes were supposed to be closed. Participants were not given any feedback on their accuracy during the experiment.

I then conducted a second experiment to study the effects of minification on verbal reports. I used a between-participant design with 13 participants in the normal condition and 14 participants in the minified condition. The procedure was the same as the first experiment except that, instead of blindly walking to the target, participants verbally indicated the distance to the target in their preferred units (yards or meters) with a resolution of at least one tenth of a unit.

4.3.3. Results and Discussion

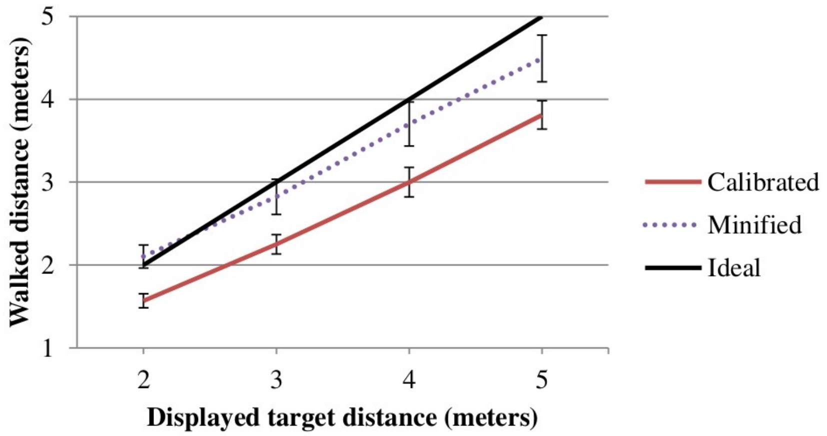


Figure 4.4. Blind walking distance judgments in a normal (calibrated) and minified virtual environment.

As shown in Figure 4.4, minification affected the distance participants walked. On average, participants walked 76% and 95% of the displayed target distance in the calibrated and minified conditions, respectively. A 2 (display condition) × 4 (target distance) repeated measures ANOVA showed that both the display condition (F (1, 23) = 6.00, $p < 0.05$) and the target distance (F (3, 23) = 277.82, $p < 0.001$) significantly affected blind walking distance judgments.

The results of the blind walking experiment were consistent with previous minification studies [28, 29] in a hallway virtual environment and showed similar results in a classroom-sized virtual environment. In addition, the magnitude of the compression in the calibrated condition was similar to other HMD direct blind walking studies [43, 36, 60, 29].

Minification changes numerous visual cues which could increase perceived distance. Minification reduces binocular disparity because the images seen in the left and right eyes are reduced in the HMD. Minification also reduces the overall optic flow for a given viewpoint movement. A familiar example of this occurs when people use binoculars which magnify an image and cause large amounts of optic flow during small amounts of movement; minification causes the opposite effect. This reduction in optic flow makes it appear that the distance to the points in the flow field have increased. Additionally, minification reduces the visual angle of any given object and therefore can cause familiar size cues to indicate that the object is farther away. Yet for all these complex phenomena, a simpler potential explanation presents itself: everything in the scene looks smaller when graphics are minified, so is it not natural that users would perceive objects in the scene as being farther away?

Figure 4.4 shows that minification deviated from the actual displayed distance as the target distance increased. This deviation could be explained by the fact that people generally pointed the center of the HMD above the nearest targets and below the furthest targets. When combined with minification (which pushes all virtual points toward the center of the screens), this may explain why the longer distances appeared compressed. Another explanation may be that participants were concerned about colliding with the virtual wall beyond the target at the longer distances. Other studies using a hallway model and different equipment have not found a similar effect [28, 29]. Because the minified and calibrated lines in Figure 4.4 are nearly parallel, a third explanation is that minification affected judged distances by a fixed amount in this experiment.

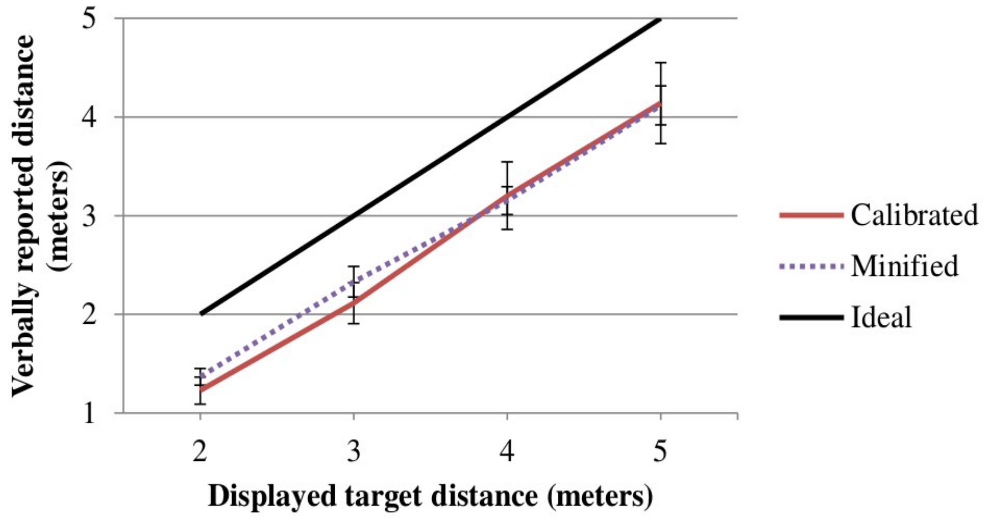


Figure 4.5. Verbal reports of distance in a normal (calibrated) and minified virtual environment.

As shown in Figure 4.5, minification did not significantly affect verbal reports of distance. On average, participants reported targets to be 74% and 77% of the veridical distance in the normal and minified conditions, respectively. A 2 (display condition) × 4 (target distance) repeated measures ANOVA showed that target distance (F (3, 24) = 204.12, p < 0.001) affected distance judgments but the display condition did not (F (1, 24) = 0.057, p = 0.81). One of the 14 participants in the minified condition was excluded from the analysis because Grubb's outlier test strongly (p < 0.01) flagged them. The flagged individual reported distances 1.6 times greater than the other participants in the condition. If the outlier is included in the analysis, the average distance indicated in the minification condition was 82% and there was still no statistically significant difference between the calibrated and minified conditions (F (1, 25) = 0.55, p = 0.46).

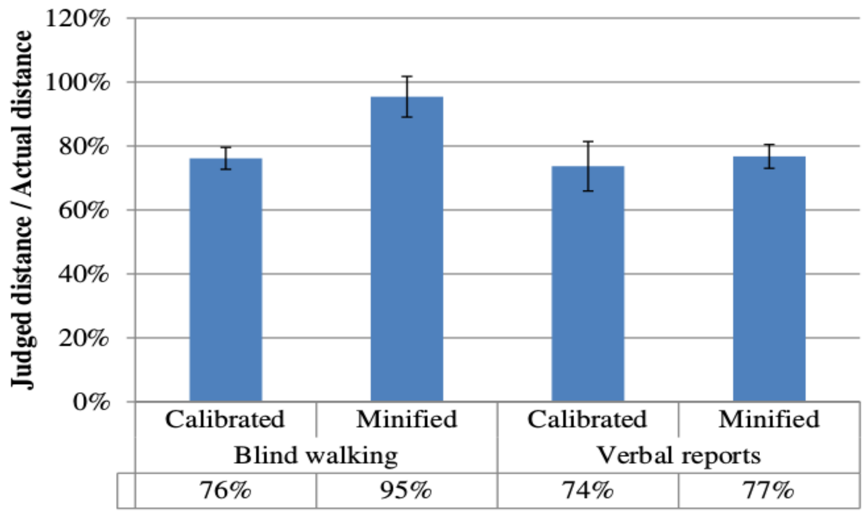


Figure 4.6. Results of blind walking and verbal report experiments.

The second experiment provided evidence that minification does not change verbal reports of distance in the same way that it changes blind walking responses in a high-fidelity virtual environment. One interpretation of the results of these experiments is that minification can be used to make blind walking in virtual environments more consistent with real environments without changing users' conscious perceptions of distance in virtual environments.

Lastly, I performed a 2 (response measure) × 2 (display condition) × 4 (target distance) repeated measures ANOVA to compare the results between the two experiments. The target distance significantly affected responses (F (3, 141) = 448.19, p < 0.001). In addition, there was a significant interaction between the response measure and the target distance (F (3, 141) = 4.31, p < 0.01) indicating different slopes of the responses shown in Figures 2 and 3. There was no statistically significant difference in response measure

(F (1, 47) = 1.97, p = 0.17) or display condition (F (1, 47) = 3.03, p = 0.09). There was no significant interaction between display condition and response measure (F (1, 47) = 1.98, p = 0.17) because the results lacked a strong crossover interaction and three of the four conditions had remarkably similar results (see Figure 4.6).

Post-hoc analysis showed that the significant interaction between response measure and target distance existed only between the two normal conditions (F (3, 69) = 3.37, p < 0.05). Additional post-hoc analysis showed that response measure did not significantly change distance judgments across the two normal conditions (F (1, 23) = 0.002, p = 0.96). However, the judged distance was significantly affected by response measure across the two minified conditions (F (1, 24) = 4.53, p < 0.05).

4.4. The Effects of Minification on Aperture Width Judgments in HMDs

4.4.1. Background

The previous study provided evidence to suggest that minification improves users' perception of distance in virtual environments (as measured by blind walking) without altering users' conscious perceptions (as measured by verbal reports). This interpretation, if correct, is a very favorable result for minification's ability to improve users' distance perception while wearing HMDs.

However, further research was needed to investigate the effects of minification on users' perceptions in virtual environments. It has already been noted that objects look smaller with minified compared to non-minified graphics. This could potentially affect users' perceptions in ways unrelated to distance. Users might judge objects as being smaller than their actual size, which could negatively impact their ability to interact with the virtual environment.

For example, what if users judged the widths of openings, such as doors, to be smaller than they actually are? Users might incorrectly conclude that they cannot pass through an opening that they in fact can pass through, or they might think that they have to turn their body to pass through an opening when in fact this is unnecessary.

Geuss et al. [18] studied the issue of user perception of aperture widths in virtual environments with normal (non-minified) graphics. They used a pair of vertical poles (see figure 4.7) and evaluated users' perception of the width between the poles using two measures. In one measure, users verbally indicated whether they believed they could walk between the poles without having to turn their body to fit (verbal report). In the other measure, users spread their hands to indicate what they believed the width of the poles to be (action-based measure). They found no statistically significant difference in users' perception in the real world compared to a virtual environment with non-minified graphics.

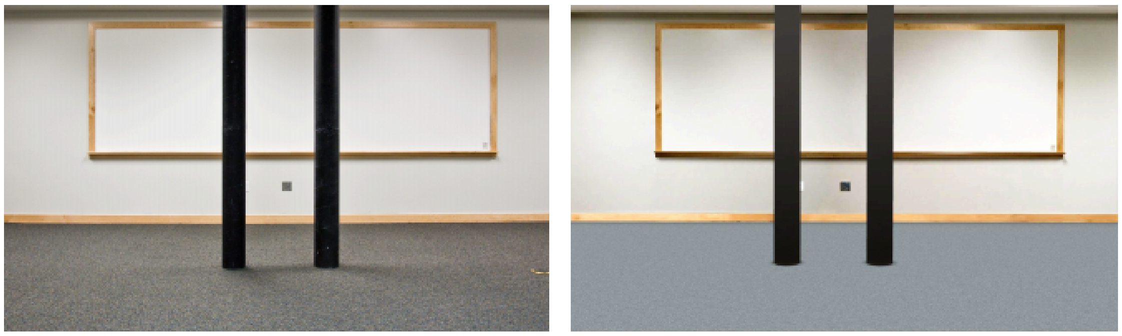


Figure 4.7. Poles used to study users' aperture width judgments in the real world (left) and virtual environment (right).

4.4.2. Study

I conducted a study [56] to evaluate users' aperture width perception by expanding on the work by Geuss et al. [18] to include minified graphics, so that I could compare users' perception in normal vs. minified graphics. I used the same hardware and virtual environment as those described in Section 4.3, except that I simplified the ceiling and floor textures in the virtual environment to reduce visual cues that users could exploit when judging aperture widths.

The study had two conditions, normal and minified graphics, with 12 participants. I used a within-subject design, with half of the participants viewing the minified condition first. There were approximately 80 trials per participant. Since I was primarily interested in whether minification affected people's conscious perceptions, I only had participants report whether they believed they could walk between the poles without having to turn their body.

For each trial, the poles were placed 3, 4, 5, or 6 meters from the viewer. The distance between the poles either began 30 cm apart and grew by 5 cm each time the participant responded in the negative, or began 60 cm apart and shrank by 5 cm each time the participant responded in the affirmative. The sequence was terminated when the participant switched their answer and maintained the switch for two consecutive trials. The order of the trials was randomized, and the software automatically tracked participants' responses so that each sequence was terminated at the appropriate time despite their order being scrambled.

4.4.3. Results and Discussion

The average point at which participants transitioned their answer was an aperture/shoulder width ratio of 1.14 (Geuss et al. [18] found 1.16). A paired t-test ($p = 0.3$) yielded no statistically significant difference between the conditions. I then binned the data into 10% gap/shoulder ratios and calculated the probability of an affirmative response for each bin (see figure 4.8). The graph also does not show a substantial difference between the conditions, as indicated by the two lines lying within each other's error bars. I observed a slight order effect which was primarily caused by a significant order effect in one participant's responses; however, this participant's data is still included in the analysis.

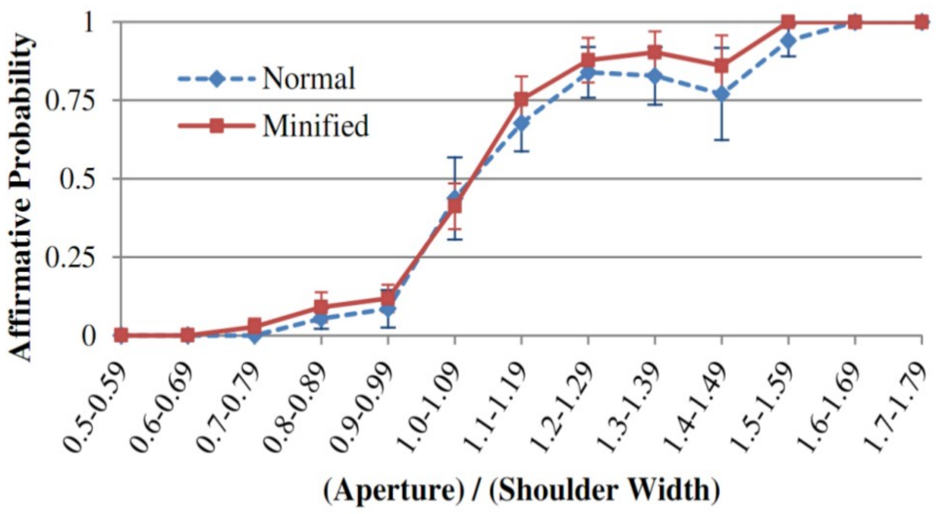


Figure 4.8. Probability of a participant switching their answer by aperture/shoulder width ratio.

This result provided further support for the hypothesis that minification does not affect users' conscious perceptions in virtual environments. Further evidence of this was provided by the fact that only two of 12 participants reported noticing anything different between the two conditions, and even these two were not sure what the difference was.

Further work would be helpful to further explore the effects of minification on user perceptions. For example, this study could be expanded to investigate users' aperture width judgments with an action-based response (such as spreading their hands). A follow-up investigation was conducted for this purpose, but that study was performed by other members of the research group and is not discussed here.

4.5. Summary

Accurate user perception in virtual environments is an important aspect of improving the usability of virtual environments, especially in training or architectural prototyping applications, but also for entertainment purposes. In the face of a mountain of research showing that users' distance judgments are compressed in virtual environments, this fact could be rather problematic.

Minification, a technique for expanding the rendered field of view and thus "zooming out" the graphics, has been shown to improve users' distance judgments in virtual environments. However, prior to the work described in this chapter, research into the effects of minification had been limited to a hallway environment and measured by blind walking. I conducted studies to evaluate the effects of minification in a room environment, to measure users' distance perception under minification with verbal reports, and to evaluate minification's effects on users' aperture width judgments.

My first study showed that minification still improves users' distance judgments in a room environment, and that it affects blind walking differently from verbal reports, with one potential (and promising) interpretation being that minification improves users' distance judgments without affecting their conscious perceptions. I conducted a follow-up study to investigate whether minification affects users' aperture width judgments as measured by verbal report and found that it does not, providing further evidence of minification’s potential usefulness as a technique to improve users’ visual perceptions in virtual environments.

Chapter 5: Summary

5.1. Summary of Experiments

My first concern regarding the usability of HMDs for everyday computing tasks was the obstacle of HMDs occluding the real world while trying to use physical input devices; i.e., keyboards. Chapter 2 describes my research into this topic, with Section 2.4 describing the main user study. For this study, I implemented a software system to assist HMD users typing on physical keyboards by utilizing a state-of-the-art touchscreen decoder adapted for physical keyboards, and a virtual keyboard assistant that highlighted which keys the user was pressing. The effectiveness of the decoder was already established, so the main hypothesis I wanted to test was whether the virtual keyboard improved users' typing accuracy. The outcome of the study was that the virtual keyboard significantly improved users' accuracy; and furthermore, that most users accomplished after-correction error rates of < 5% even with the backspace and delete keys disabled.

This work provided evidence for the feasibility of using a physical keyboard while wearing an HMD, and also demonstrated that a camera-less system can be helpful for users in this situation. This is an important finding since mixed-reality solutions that superimpose parts of the real world within the virtual environment rely on external cameras that users might not have the budget or the space for.

Based on these results, I developed a prototype virtual reality window manager called Fish Out of Water (FOW) which displays application windows in a cylindrical virtual environment with 360 degrees of space, which the user can view simply by turning his or her head. FOW also utilizes the virtual keyboard developed for the typing study described previously. The development of this system is described in detail in Chapter 3. I designed a task wherein users had to compose an email for a fictional supervisor by gathering data from many different application windows. I conducted a user experience survey in which users performed this task using FOW, tracked some usage metrics, and also gathered qualitative user feedback. This study is described in Section 3.4. All users were able to complete the task with an average time of 30 minutes, with most achieving 99% accuracy. In qualitative feedback, users praised the virtual keyboard's ability to aid their typing and the convenience of a 360-degree workspace for arranging windows. Conversely, users criticized the blurriness at the edges of the lenses (due to chromatic aberration) and some common features missing due to FOW being a prototype and not a finished product.

The work with FOW provided strong evidence for the potential of using HMDs with everyday computing tasks. However, it also exposed obstacles to realizing this vision. In particular, HMD display and lens technology may need to provide extremely high resolution and clarity at affordable cost in order for this use of HMDs to be feasible. User comments on FOW provided useful feedback for improvements that could be made to future virtual reality window managers.

Chapter 4 describes work that I have conducted regarding distance compression in HMDs and a compensation technique called minification. I conducted two user studies, described in sections 4.3.2 and 4.4.2, investigating the effects of minification on users' perception in HMDs. I found that minification helped users to judge distances more accurately and looked for, but did not find, evidence of undesirable side effects of minification, specifically user judgments of aperture widths. These were promising results for the effectiveness of minification, and have also yielded some interesting insights into the human perceptual system.

Since this work, it has been found that distance judgments are more accurate in HMDs with a wide field of view. Additionally, recent work by Li et al [32] has taken a different approach and found that, in HMDs with a narrow field of view, a bright frame can also improve distance perception.

Appendix A describes a sizable body of work I have accumulated which is not directly related to the central theme of my research. This includes a user study on FlowTour, an application for visualizing 3D flow fields; a Distributed Graphics Renderer I developed for synchronizing graphical rendering on a display wall, which was used in subsequent research; iISoP, the immersive Interactive Sonification Platform; and RobotRun, training software for using and programming industrial robotic arms.

5.2. The Future

Despite the large body of virtual reality research and the proliferation of consumer-affordable HMDs, this field remains in its infancy. The true potential of VR may, as yet, remain largely untapped. Entertainment applications for VR are seeing substantial commercial development, but this technology has so much more to offer than immersive video games. Can virtual reality be leveraged to enable novel or more efficient uses for everyday productivity applications like Web browsing and email? I have presented work that takes some small steps towards answering this question.

Based on my research, the use of virtual reality for enhancing everyday computing tasks appears worthy of continued investigation. Many challenges remain, including technological limitations of the physical displays in HMDs themselves. Yet the potential is clear to see. Hopefully this work will serve as a stepping stone for the continued development of software that helps virtual reality realize its potential as a home platform for more than entertainment purposes.

References

[1] Andre, J., & Rogers, S. (2006). Using verbal and blind-walking distance estimates to investigate the two visual systems hypothesis. *Perception & Psychophysics*, *68*(3), 353-361.

[2] Boulanger, R., (2000). *The Csound Book*. MIT Press.

[3] Bowman, D. A., Rhoton, C. J., & Pinho, M. S. (2002, September). Text input techniques for immersive virtual environments: An empirical comparison. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* (Vol. 46, No. 26, pp. 2154-2158). Sage CA: Los Angeles, CA: SAGE Publications.

[4] Bowman, D. A., & Wingrave, C. A. (2001, March). Design and evaluation of menu systems for immersive virtual environments. In *Proceedings of Virtual Reality, 2001* (pp. 149-156). IEEE.

[5] Budhiraja, P., Sodhi, R., Jones, B., Karsch, K., Bailey, B., & Forsyth, D. (2015). Where's My Drink? Enabling Peripheral Real World Interactions While Using HMDs. *arXiv preprint arXiv:1502.04744*.

[6] Clark, J. (1997, December). Comparison of SGML and XML. Retrieved from https://www.w3.org/TR/NOTE-sgml-xml

[7] Coombs, J.H., Renear, A.H., & DeRose, S.J. (1987). Markup systems and the future of scholarly text processing. *Communications of the ACM 30.11,* pp. 933-947. ACM.

[8] Creem-Regehr, S. H., Willemsen, P., Gooch, A. A., & Thompson, W. B. (2005). The influence of restricted viewing conditions on egocentric distance perception: Implications for real and virtual indoor environments. *Perception*, *34*(2), 191-204.

[9] Czerwinski, M., Robertson, G., Meyers, B., Smith, G., Robbins, D., & Tan, D. (2006, April). Large display research overview. In *CHI'06 extended abstracts on Human factors in computing systems* (pp. 69-74). ACM.

[10] Czerwinski, M., Smith, G., Regan, T., Meyers, B., Robertson, G., & Starkweather, G. (2003). Toward characterizing the productivity benefits of very large displays. In *Proceedings of Interact 2003* (Vol. 3, pp. 9-16).

[11] DiVerdi, S. (2007). Towards Anywhere Augmentation. (Doctoral dissertation). ProQuest.

[12] DiVerdi, S., Nurmi, D., & Höllerer, T. (2003, October). ARWin-a desktop augmented reality window manager. In *Proceedings of the 2nd IEEE/ACM International Symposium on Mixed and Augmented Reality* (p. 298). IEEE Computer Society.

[13] Elmqvist, N. (2003). 3Dwm: A platform for research and development of three-dimensional user interfaces. *Technical Report CS: 2003-04, Chalmers Department of Computing Science.*

[14] Ens, B. M., Finnegan, R., & Irani, P. P. (2014, April). The personal cockpit: a spatial interface for effective task switching on head-worn displays. In *Proceedings of the 32nd annual ACM conference on Human factors in computing systems* (pp. 3171-3180). ACM.

[15] Ens, B., Hincapié-Ramos, J. D., & Irani, P. (2014, October). Ethereal planes: a design framework for 2D information space in 3D mixed reality environments. In *Proceedings of the 2nd ACM symposium on Spatial user interaction* (pp. 2-12). ACM.

[16] Feiner, S., MacIntyre, B., Haupt, M., & Solomon, E. (1993, December). Windows on the world: 2D windows for 3D augmented reality. In *Proceedings of the 6th annual ACM symposium on User interface software and technology* (pp. 145-155). ACM.

[17] Fennell, P. (2013). Extremes of XML. *XML LONDON 2013.*

[18] Geuss, M., Stefanucci, J., Creem-Regehr, S., & Thompson, W. B. (2010, July). Can I pass?: using affordances to measure perceived size in virtual environments. In *Proceedings of the 7th Symposium on Applied Perception in Graphics and Visualization* (pp. 61-64). ACM.

[19] Grechkin, T. Y., Nguyen, T. D., Plumert, J. M., Cremer, J. F., & Kearney, J. K. (2010). How does presentation method and measurement protocol affect distance estimation in real and virtual environments?. *ACM Transactions on Applied Perception (TAP), 7*(4), 26.

[20] Gebhardt, S., Pick, S., Leithold, F., Hentschel, B., & Kuhlen, T. (2013). Extended pie menus for immersive virtual environments. *IEEE Transactions on Visualization and Computer Graphics,* 19(4), 644-651.

[21] Ha, T., Feiner, S., & Woo, W. (2014, September). WeARHand: Head-worn, RGB-D camera-based, bare-hand user interface with visually enhanced depth perception. In *2014 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)* (pp. 219-228). IEEE.

[22] Hutchings, D. R., Smith, G., Meyers, B., Czerwinski, M., & Robertson, G. (2004, May). Display space usage and window management operation comparisons between single monitor and multiple monitor users. In *Proceedings of the working conference on Advanced visual interfaces* (pp. 32-39). ACM.

[23] Jeon, M., Landry, S., Ryan, D. J., & Walker, J. W. (2015). Technologies expand aesthetic dimensions: Visualization and sonification of embodied penwald drawings. In A. L. Brooks, E. Ayiter, and O. Yazicigil (Eds.), *Arts and Technology. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering (LNICST) vol. 145.* (pp. 69-76). Springer International Publishing.

[24] Jeon, M. Smith, M. T., Walker, J. W., & Kuhl, A. S. (2014). Constructing the immersive interactive sonification platform (iISoP). In N. Streitz and P. Markopoulos (Eds.): Distributed, Ambient, and Pervasive Interactions (DAPI) 2014, *Lecture Notes in Computer Science (LNCS) 8530*, pp. 337-348. Springer International Publishing Switzerland.

[25] Jones, J. A., Krum, D. M., & Bolas, M. T. (2016). Vertical Field-of-View Extension and Walking Characteristics in Head-Worn Virtual Environments. *ACM Transactions on Applied Perception (TAP)*, *14*(2), 9.

[26] Knapp, J. M., & Loomis, J. M. (2004). Limited field of view of head-mounted displays is not the cause of distance underestimation in virtual environments. *Presence: Teleoperators and Virtual Environments*, *13*(5), 572-577.

[27] Kuhl, S.A., Sergeyev, A., Walker, J., Lakshmikanth, S.A., Highum, M., Alaraje, N., Zhang, R. & Kinney, M.B. (2016, June). Enabling Affordable Industrial Robotics Education through Simulation. *ASEE Annual Conference & Exposition.* New Orleans, Louisiana.

[28] Kuhl, S. A., Thompson, W. B., & Creem-Regehr, S. H. (2006, July). Minification influences spatial judgments in virtual environments. In *Proceedings of the 3rd symposium on Applied perception in graphics and visualization* (pp. 15-19). ACM.

[29] Kuhl, S. A., Thompson, W. B., & Creem-Regehr, S. H. (2009). HMD calibration and its effects on distance judgments. *ACM Transactions on Applied Perception (TAP), 6*(3), 19.

[30] Kunz, B. R., Wouters, L., Smith, D., Thompson, W. B., & Creem-Regehr, S. H. (2009). Revisiting the effect of quality of graphics on distance judgments in virtual environments: A comparison of verbal reports and blind walking. *Attention, Perception, & Psychophysics*, *71*(6), 1284-1293.

[31] Lee, M., & Woo, W. (2003, December). ARKB: 3D vision-based Augmented Reality Keyboard. In *ICAT*.

[32] Li, B., Nordman, A., Walker, J., & Kuhl, S.A. (2016). The effects of artificially reduced field of view and peripheral frame stimulation on distance judgments in HMDs. In *Proceedings of the ACM Symposium on Applied Perception*. ACM, 53–56.

[33] Loomis, J. M., Da Silva, J. A., Fujita, N., & Fukusima, S. S. (1992). Visual space perception and visually directed action. *Journal of Experimental Psychology: Human Perception and Performance, 18*(4), 906.

[34] Ma, J., Walker, J., Wang, C., Kuhl, S., & Shene, C. K. (2014, March). FlowTour: An automatic guide for exploring internal flow features. In *Visualization Symposium (PacificVis), 2014 IEEE Pacific* (pp. 25-32). IEEE.

[35] McGill, M., Boland, D., Murray-Smith, R., & Brewster, S. (2015, April). A dose of reality: overcoming usability challenges in vr head-mounted displays. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (pp. 2143-2152). ACM.

[36] Mohler, B. J., Creem-Regehr, S. H., & Thompson, W. B. (2006, July). The influence of feedback on egocentric distance judgments in real and virtual environments. In *Proceedings of the 3rd symposium on applied perception in graphics and visualization* (pp. 9-14). ACM.

[37] Nan, X., Zhang, Z., Zhang, N., Guo, F., He, Y., & Guan, L. (2014). vDesign: a CAVE-based virtual design environment using hand interactions. *Journal on Multimodal User Interfaces,* 8(4), 367-379.

[38] Processing. https://processing.org/

[39] Reiling, F. (2014). Toward General Purpose 3D User Interfaces: Extending Windowing Systems to Three Dimensions. (Master's Thesis). Retrieved from https://github.com/evil0sheep/MastersThesis/blob/master/thesis.pdf?raw=true

[40] Rieser, J. J., Ashmead, D. H., Talor, C. R., & Youngquist, G. A. (1990). Visual perception and the guidance of locomotion without vision to previously seen targets. *Perception*, *19*(5), 675-689.

[41] Robertson, G., Czerwinski, M., Baudisch, P., Meyers, B., Robbins, D., Smith, G., & Tan, D. (2005). The large-display user experience. *Computer Graphics and Applications*, IEEE, 25(4), 44-51.

[42] Robertson, G., Van Dantzich, M., Robbins, D., Czerwinski, M., Hinckley, K., Risden, K., Thiel, D., & Gorokhovsky, V. (2000, April). The Task Gallery: a 3D window manager. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 494-501). ACM.

[43] Sahm, C. S., Creem-Regehr, S. H., Thompson, W. B., & Willemsen, P. (2005). Throwing versus walking as indicators of distance perception in similar real and virtual environments. *ACM Transactions on Applied Perception (TAP)*, *2*(1), 35-45.

[44] Scheirer, E. D. (1998, May). The MPEG-4 structured audio standard. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on* (Vol. 6, pp. 3801-3804). IEEE.

[45] Seward, Z.M. (June, 2014). Virtual reality headset Oculus Rift meets the Bloomberg terminal. *Quartz.* http://qz.com/218129/virtual-reality-headset-oculus-rift-meets-the-bloomberg-terminal/

[46] Simmons, T., & Manahan, M. (1999, January). The effects of monitor size on user performance and preference. In *Human Factors and Ergonomics Society Annual Meeting Proceedings* (Vol. 43, No. 24, pp. 1393-1393).

[47] Thompson, W. B., Willemsen, P., Gooch, A. A., Creem-Regehr, S. H., Loomis, J. M., & Beall, A. C. (2004). Does the quality of the computer graphics matter when judging distances in visually immersive environments? *Presence*, *13*(5), 560-571.

[48] Topol, A. (2000, April). Immersion of Xwindow applications into a 3D workbench. In *CHI'00 extended abstracts on Human factors in computing systems* (pp. 355-356). ACM.

[49] Vertanen, K., & Kristensson, P. O. (2011, August). A versatile dataset for text entry evaluations based on genuine mobile emails. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services* (pp. 295-298). ACM.

[50] Vertanen, K., Memmi, H., Emge, J., Reyal, S., & Kristensson, P. O. (2015, April). VelociTap: Investigating fast mobile text entry using sentence-based decoding of touchscreen keyboard input. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (pp. 659-668). ACM.

[51] Vertanen, K., Memmi, H., & Kristensson, P. O. (2013, October). The feasibility of eyes-free touchscreen keyboard typing. In *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility* (p. 69). ACM.

[52] Virtual Desktop on Steam. http://store.steampowered.com/app/382110

[53] Walker, J., Kuhl, S. A., & Vertanen, K. (2016). Decoder-Assisted Typing Using an HMD and a Physical Keyboard. *CHI '16: Extended Abstracts of the ACM International Conference on Human Factors in Computing Systems (CHI 2016 workshop).* ACM.

[54] Walker, J., Li, B., Kuhl, S.A., & Vertanen, K. (2017, May). Efficient Typing on a Visually Occluded Physical Keyboard. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* ACM.

[55] Walker, J., Smith, M. T., & Jeon, M. (2015). Interactive sonification markup language (ISML) for efficient motion-sound mappings. In M. Kurosu (Ed.). *Human-Computer Interaction: Interaction Technologies, Part II, HCII 2015, LNCS 9170,* pp. 1-10. Springer International Publishing Switzerland.

[56] Walker, J., Zhang, R., & Kuhl, S. A. (2012, August). Minification and gap affordances in head-mounted displays. In *Proceedings of the ACM Symposium on Applied Perception* (p. 124). ACM.

[57] Wang, G. (2008). The ChucK Audio Programming Language (Doctoral dissertation). Retrieved from http://www.cs.princeton.edu/~gewang/thesis.pdf

[58] Wayland. http://wayland.freedesktop.org/

[59] Wayland 3D Compositor on Oculus Rift /r/linux (Wayland 3D Compositor on Oculus Rift : linux) http://www.reddit.com/r/linux/comments/287nup/wayland_58_compositor_on_oculus_rift/

[60] Willemsen, P., Colton, M. B., Creem-Regehr, S. H., & Thompson, W. B. (2009). The effects of head-mounted display mechanical properties and field of view on distance judgments in virtual environments. *ACM Transactions on Applied Perception (TAP), 6*(2), 8.

[61] Willemsen, P., Gooch, A. A., Thompson, W. B., & Creem-Regehr, S. H. (2008). Effects of stereo viewing conditions on distance perception in virtual environments. *Presence: Teleoperators and Virtual Environments, 17*(1), 91-101.

[62] Witmer, B. G., & Kline, P. B. (1998). Judging perceived and traversed distance in virtual environments. *Presence, 7*(2), 144-167.

[63] Witmer, B. G., & Sadowski, W. J. (1998). Nonvisually guided locomotion to a previously viewed target in real and virtual environments. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, *40*(3), 478-488.

[64] World Wide Web Consortium. (2008, November). Extensible Markup Language (XML) 1.0 (Fifth Edition). Retrieved from https://www.w3.org/TR/REC-xml/

[65] Wu, B., Ooi, T. L., & He, Z. J. (2004). Perceiving distance accurately by a directional process of integrating ground information. *Nature*, *428*(6978), 73-77.

[66] Yi, X., Yu, C., Zhang, M., Gao, S., Sun, K., & Shi, Y. (2015, November). Atk: Enabling ten-finger freehand typing in air based on 58 hand tracking data. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology* (pp. 539-548). ACM.

[67] Zhang, R., Nordman, A., Walker, J., & Kuhl, S. A. (2012). Minification affects verbal-and action-based distance judgments differently in head-mounted displays. *ACM Transactions on Applied Perception (TAP)*, *9*(3), 14.

Appendix A: Anything and Everything: The Eclectic

I also conducted a significant amount of research that does not directly relate to the central theme of usability in virtual environments. For completeness, this other research is summarized here.

A.1. FlowTour

A.1.1. Background

Michigan Tech's computer graphics research group created a flow field visualization application called FlowTour [34]. A flow field is a representation of the flow of fluids through a space. FlowTour renders 3D graphical depictions of flow fields. The application's novel contribution is that it automatically identifies critical regions, selects good viewpoints for viewing those regions, and generates a camera path through the flow field, providing a "tour" of the most interesting features in the field. Intuitively, a critical region is an area in a flow field which contains meaningful features such as sources, sinks, and vortices, contrasted with regions composed of chaotic turbulence.

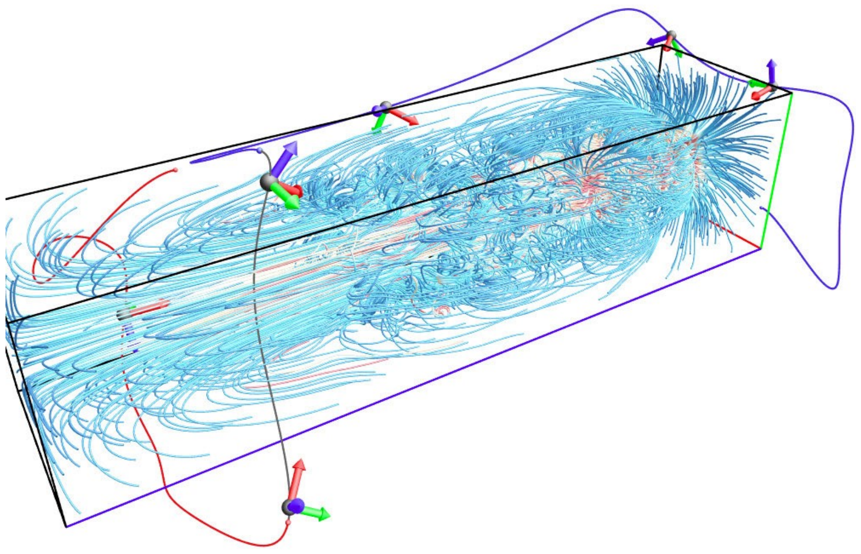


Figure A.1. Screen capture of a solar plume flow field rendered by FlowTour, and the accompanying view path that was generated.

A.1.2. Study

I designed and executed a user study to evaluate the effectiveness of FlowTour. I used a design of 3 conditions (FlowTour path, randomly-generated internal path, and external path) × 2 tasks (answer questions and identify critical regions). The random internal path was constrained to be the same length with the same amount of total rotation as the FlowTour path. The external path was calculated using a strategy similar to FlowTour, with skeleton-based seeding, but constrained to remain outside of the data set. I recruited 10 users for the FlowTour condition, 11 users for the random condition, and 10 users for the external condition. Participants were either recruited from the university subject pool or were paid $10 for participating in the study. All participants were recruited from the university campus and the local community.

For each experimental session, users were shown seven flow field data sets, one for practice and six for evaluation. After each tour, the users were asked several multiple-choice questions about features in the flow field, and were then asked to identify as many critical regions as they could find within a set time limit. I hypothesized that users would perform better on both tasks with FlowTour.

First, users were given a briefing explaining the basic characteristics of flow fields, critical regions they would need to recognize, and the tasks they would be expected to perform. After the briefing, the users were given the chance to practice their tasks on one data set before beginning the main study. The practice session took the users through one complete set of tasks and gave them the opportunity to become familiar with the program interface and working with flow fields. After the users finished the practice set, the main study commenced, consisting of six data sets. At this time, users were only allowed to ask the researchers for clarification about the meaning of specific questions or how to use the program.

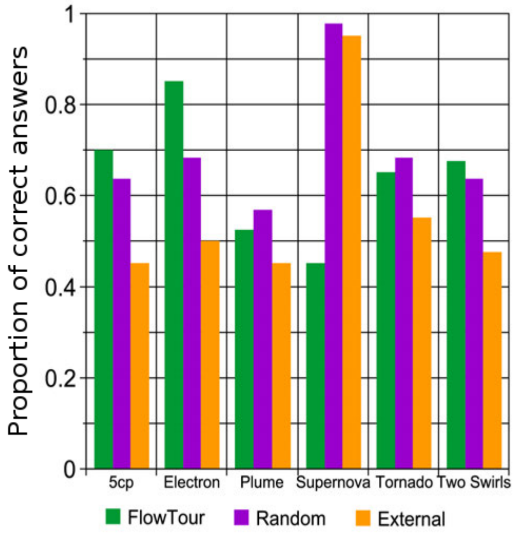
The procedure for each data set was as follows. The user was shown an animation of the complete path through the data set. The speed of the animation could be adjusted if desired. After the animation, the user was presented with several questions and then asked to identify critical regions in the data set. The user had a limited time to perform these tasks. While performing these tasks, the user could revisit any part of the path using a slider. This functionality was useful for answering questions and was required for identifying critical regions.

Users completed all seven data sets in one sitting. They were not allowed to take breaks or leave the workstation until the experiment was completed. Users could opt to terminate the experiment at any time, but none chose to do so. The entire experiment took approximately 60 minutes for most users, including initial paperwork, briefing, and post-experiment questionnaire. The questionnaire asked the users for subjective feedback and suggestions for improvement regarding the experiment and the program's user interface.

A.1.3. Results and Discussion

I present the study results in two aspects: user accuracy on multiple-choice questions and the proportion of critical regions correctly identified. Because user performance varied widely by data set, each data set was analyzed individually, comparing user performance by path (FlowTour, random internal, and external). I used one-way ANOVA to analyze statistical significance between the conditions with a standard significance level $\alpha = 0.05$.

Multiple-choice questions Each data set was analyzed individually by comparing users' average proportion of correct answers by different paths. The results are given in Figure A.2. Performance differences in this condition were not statistically significant except for the supernova data set, $p < 0.0001$, wherein the random and external paths substantially outperformed the FlowTour path.



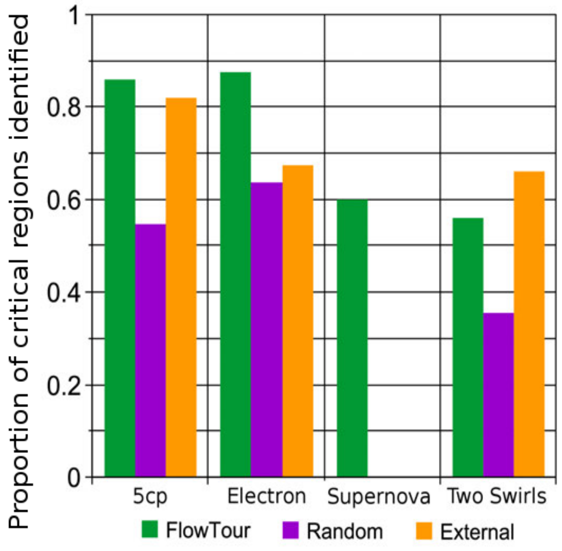


Figure A.2. Average proportion of correct answers to multiple-choice questions by data set (left) and proportion of critical regions identified by data set (right). For the supernova data set, the proportion of critical regions correctly identified in the random and external conditions is zero.

Critical regions identification Similar to the multiple-choice questions, each data set was analyzed individually. The analysis was done by comparing how many critical regions the user correctly identified against the total number of critical regions in the data set, such that a value of 1.0 indicates the user found every critical region. The results are given in Figure A.2. The solar plume and tornado data sets were excluded from this analysis, the former because it is a turbulent mass that lacks coherent critical regions, the latter because it contains one enormous critical region that is easily identifiable by all users in all conditions. In all cases, users correctly identified more critical regions with the FlowTour path, except for the two swirls data set where the external path slightly outperformed the FlowTour path. All of these results are statistically significant ($p < 0.05$).

Discussion The overall result of the experiment is that, in the multiple-choice questions condition, users performed better with the FlowTour path and the random internal path over the external path, albeit without statistical significance in most cases. Conversely, on most data sets, users performed better at identifying critical regions with the FlowTour path. One exception to this pattern is that users answered questions about the supernova data set more accurately with the random and external paths. This might be due to the fact that the supernova contains only a single critical region surrounded by chaotic turbulence, which hinders the FlowTour algorithm from plotting an effective path through the data set.

The multiple-choice questions were designed to evaluate users' overall knowledge of the general characteristics of each data set. Lack of statistical significance hinders me from drawing definitive conclusions, but the fact that both the FlowTour path and the random internal path outperformed the external path in all but one case suggests that being able to view the interior of a data set may indeed provide users with better overall knowledge of that data set. Additionally, the relatively good performance of the random path in this task suggests that getting a general "feel" for a data set may not require a sophisticated approach. Conversely, users performed better at identifying critical regions with the FlowTour path. This result is not surprising since exposing critical regions is a major objective of the FlowTour algorithm. These results demonstrate that finding and recognizing critical regions—and by extension, other kinds of specific flow patterns—is enhanced by an algorithmic approach designed to aid this task, and that a random approach is not adequate to ensure optimal performance.

I conclude that FlowTour is not significantly more useful than a haphazard approach when gathering general information about flow fields, but FlowTour is effective in aiding users to identify critical regions, especially hidden or occluded flow features.

A.2. Distributed Graphics Renderer

A.2.1. Background

Michigan Tech’s Immersive Visualization Studio (IVS) lab contains a display wall of (3 × 8) 24 monitors. These monitors are run on a computational cluster consisting of one head node and eight tail nodes, each of which is responsible for rendering to three monitors. Therefore, in order to render on the entire display wall, any program running on the IVS system must have a way of rendering the appropriate parts of its scene on each of the tail nodes while keeping them synchronized with each other.

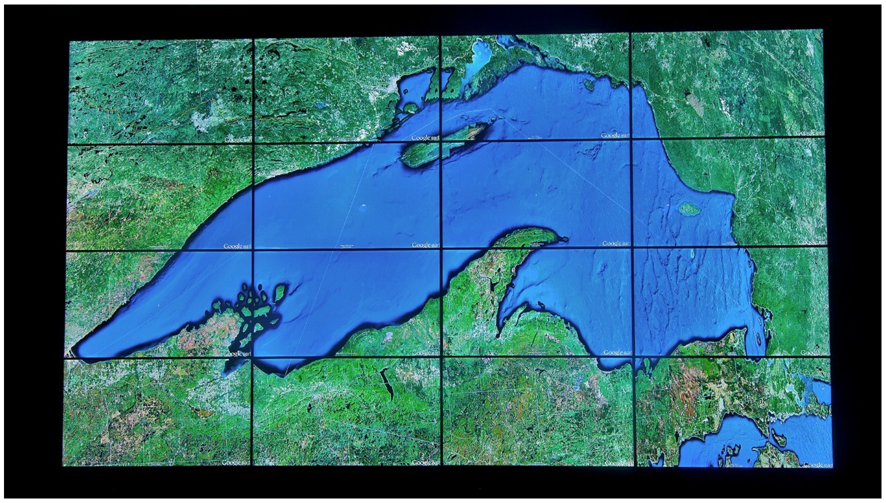


Figure A.3. The IVS display wall.

Initially, the only software available to perform this function was Chromium. Chromium accepts an OpenGL application as an argument and automatically takes care of distributing and synchronizing the rendering by forwarding OpenGL calls. However, hands-on experience with the system revealed that Chromium's performance was unacceptably slow for complex applications. A more efficient alternative needed to be developed. I was tasked with developing that alternative.

A.2.2. Software

To address these requirements, I developed the Distributed Graphics Renderer, or DGR. The initial version of DGR was designed as a template application: a minimalistic OpenGL program that rendered its graphics on the display wall. The intent was that developers could use DGR as a basis and modify the source to display the desired scene.

DGR’s design is as follows. The user runs eight copies of the same application, one on each tail node. Each process is run with a different display frustum corresponding to which monitors on the display wall that node is responsible for rendering on. I refer to these as the *slave* processes. At the same time, the user runs a *relay* on the head node, and a *controller*.

The controller is responsible for generating state information which is sent to the relay via UDP packets. This state information can be user-controlled (i.e., responding to mouse and keyboard input) or generated programmatically (e.g., simulations). The controller can be another copy of the application to be rendered, but this is not required; all that is necessary is that it send the correct state information. The relay then forwards all of this state information to each of the slave processes via UDP broadcast.

The state information consists of all the data that the application requires in order to render the scene. This consists of all the data that cannot be programmatically derived from the initial state; most commonly, input from a human user. This data is not expressed as a delta from the previous state; rather, every packet (or sequence of packets) from the controller comprises a set of complete state information.

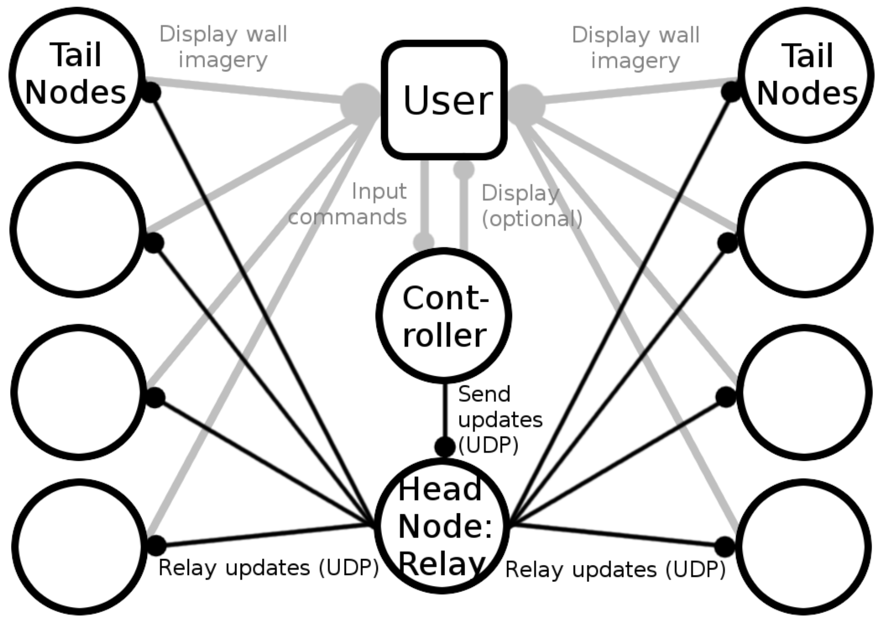


Figure A.4. Diagram of DGR's design.

Note that this design takes advantage of the particular characteristics of this problem to enable good performance and guaranteed synchronization while remaining simple in both the conception and implementation. Because the display cluster is connected by a high-performance LAN, there is very little worry of packet loss. Therefore, DGR can send all of the necessary data via UDP packets without requiring any overhead to guarantee delivery. Even if a packet is dropped here and there, synchronization is still guaranteed because every packet comprises complete state information and packets are broadcast simultaneously from the head node. There is no risk that errors might accumulate, causing a particular node might get out of sync with other nodes due to dropping packets as would be the case if state information were sent as a delta. Finally, the communication is entirely one-way (controller → relay → slaves) which makes the communication model very simple. The design is summarized in Figure A.4.

Despite these desirable characteristics, some developers were reluctant to adopt the initial version of DGR. This was due to its implementation as a template application, which was likely seen as inconvenient to use. Subsequently, DGR was modified to be implemented as a simple C library. In order to use the DGR library, the developer must first initialize the library by calling `dgr_init()`. Once DGR has been initialized, variables that comprise state information to be sent are added with `dgr_setget()`. Finally, the state is sent (by the controller) or received (by the slaves) by calling `dgr_update()`. The application is configured as either a controller or a slave at compilation, and the functions will automatically do the right thing based on this setting. Finally, call `dgr_exit()` from the controller to send a kill signal to the slave processes so that they will not continue to run on the tail nodes after the controller has terminated.

Of course, there is a possibility that the controller might terminate abnormally, which would prevent the kill signal from being sent to the slaves. DGR accounts for this as well. The slaves regularly check to see if they are receiving signals from the controller. If a sufficient amount of time passes without receiving any signals, the slaves automatically terminate themselves.

DGR has proven to be fast and reliable, with no bugs reported as of this writing. It has been used in numerous research projects, including the Mind Music Machine Lab’s Immersive Interactive Sonification Platform, which is discussed next.

A.3. The Immersive Interactive Sonification Platform

A.3.1. Background

Besides graphics, audio is another important component of human-computer interaction. Speech is the most common way to convey information to users through audio channels, but it is possible to utilize other kinds of sound as well. The use of non-speech audio to convey information is known as sonification. When this audio provides useful information for refining the user's commands and the user's interaction likewise influences the audio, the resulting feedback loop is known as interactive sonification.

In addition to its display wall, the IVS lab also has a Vicon tracking system which can track the locations of special reflective markers using 12 infrared cameras. Michigan Tech's Mind Music Machine (Tri-M) lab launched an initiative to take advantage of the features of this room to construct a sonification platform known as the immersive Interactive Sonification Platform (iISoP). Although focused on interactive sonification, iISoP also includes significant graphical components. I participated in the development of graphical software for iISoP.

A.3.2. Constructing iISoP

iISoP was initially developed through a series of three phases of increasing complexity [24]. The first phase was designed as an interactive map which would play recordings in different languages based on the user's position. The second was rendered an enormous instrument on the display wall, which the user could play through their position and body motions. The third phase made the jump to a robust platform for interactive sonification.

Table A.1. Comparisons of sonifications among different development phases

	Phase I Interactive map	Phase II Big instruments	Phase III Interactive sonification
Tracking	Location of the wand	Movement of user's ankles or wrists	Location, movement, and gestures of user's whole body
Sound Generated	Pre-recorded speech	General MIDI (frequency)	Virtual instruments (e.g., Korg Legacy Cell)
Sonification Information	Same across languages	Musical frequency or type of instrument	Multiple sound profiles
Sonification Mapping	Horizontal Location: Language Distance to Map: Amplitude Wand Flip: Gender of voice	Horizontal Location: Musical scale Distance to Map: Whole (white)/semi (black) key Jumping Height: Velocity	Many-to-many mappings
Visualization	Distance to Map: Zoom	Horizontal Location: Highlighting of the keyboard	Many-to-many mappings
Level of Interactivity	Reactivity	Reactivity	Full interactivity

Interactive Map As a test bed, the first phase of iISoP was developed as an interactive map (see Figure A.5). The tracked object in this case was a wand held by the user. With this map, users could listen to a pre-recorded introduction to Michigan Tech in different languages (e.g., American English, British English, Hindi, Chinese, Japanese, Korean, German, French, Spanish, Persian, etc.) based on their location in front of the display wall. Depending on the user's distance from the map, the amplitude of the voice clip would change (e.g., 6dB increase when the user approaches the map by one unit), accompanied by the visualization zooming in on the appropriate location. Depending on the user's horizontal location, the recording was panned using a 5.1 speaker system. The wand could be flipped upside-down to change the gender of the voice reading the pre-recorded greeting.

This first stage had a discrete one-to-one mapping between the user's location and the speech sounds. This was properly a "responsive" rather than interactive system, since communication was entirely one-way. This phase paved the way for the more sophisticated systems that followed.

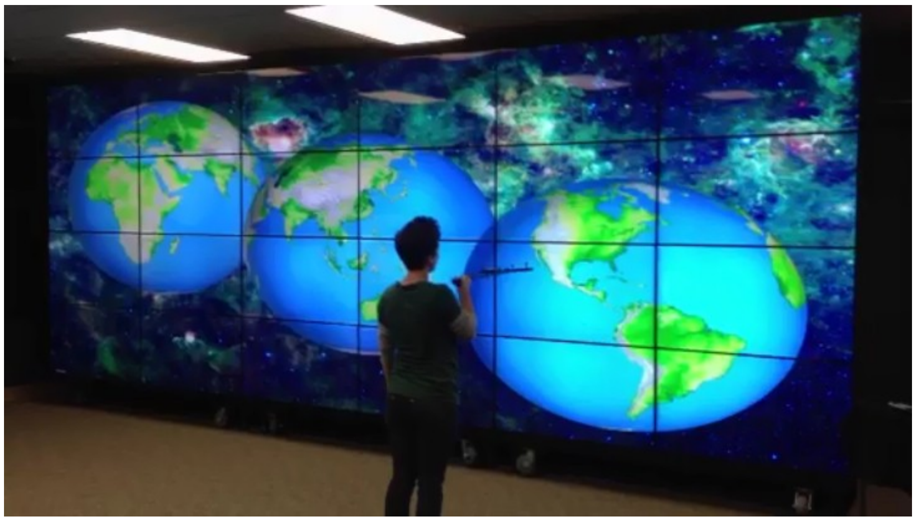


Figure A.5. Users can listen to an introduction to Michigan Tech in different languages based on their location relative to the world map.

Big Instruments Next, I implemented big virtual instruments (e.g., keyboard, percussions) that responded to the users' movements. For this system, users' feet were tracked by attaching tracking markers to their ankles. Figure A.6 shows a user playing a big piano on the display wall. The sound was generated using general MIDI with different velocities depending on how high the user jumped. The corresponding key was highlighted on the screen depending on the user's horizontal location.

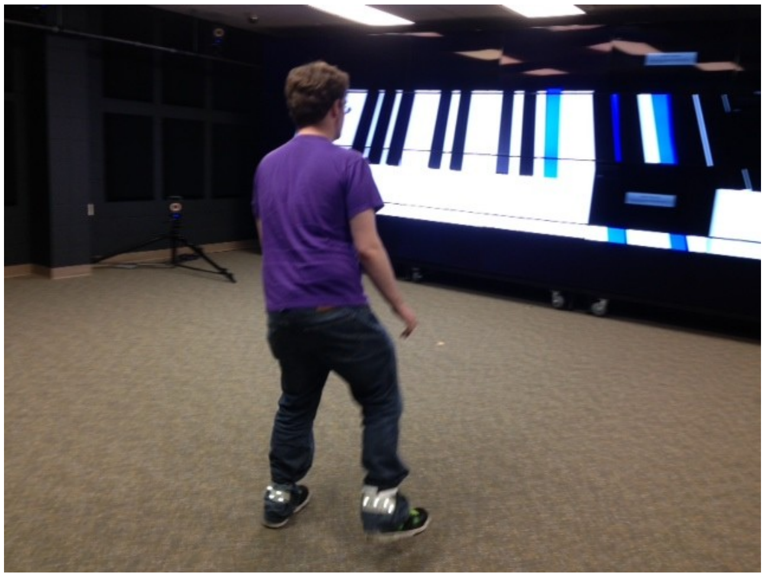


Figure A.6. Users can play big instruments with movements in front of the display wall.

In this phase, the sonification system generated real-time sounds using MIDI, contrasted with the pre-recorded files used in the first phase. In addition, each location contained musically different semantics, enabling continuous variables, unlike the interactive map.

Interactive Sonification For more advanced design research, I developed a fine-tuned interactive sonification system in the third phase. In contrast to one-on-one mapping used in the previous two phases, this phase allowed for many-to-many mappings. In this phase, users could wear a number of trackable objects simultaneously. The positions of these objects were matched with multiple sonification and visualization parameters. For researchers without the programming background to configure these mappings, I created a specialized scripting language, which is described in detail in section A.3.3. In addition to real-time responses to user input, the system also logged elapsed time, average velocity, average acceleration, and proximity to other objects, which could be used to further tune the audio and visual feedback. This more sophisticated data tracking improved the system's interactivity. This system was usable not only as a scientific research platform, but also as an experimental artistic performance research program, which I explore more in section A.3.4.

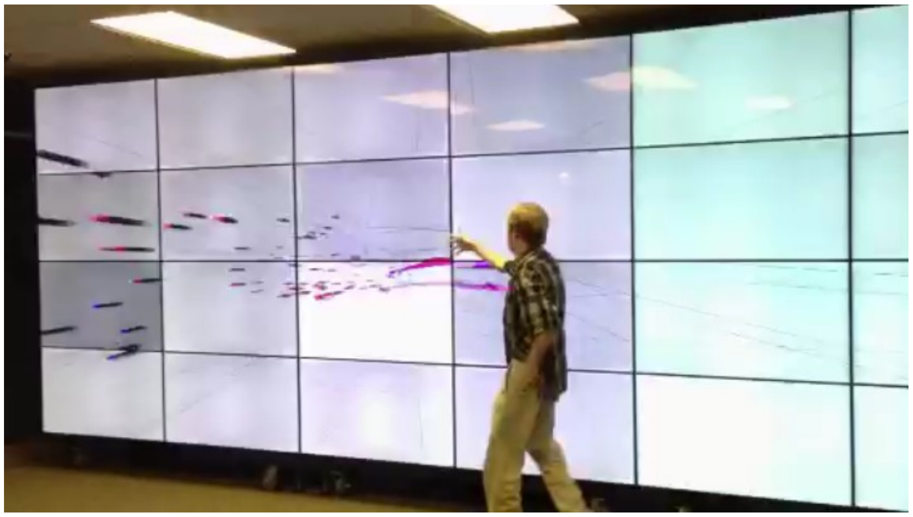


Figure A.7. Users can improvise real-time sonification and visualizations based on their location, movements, and gestures. They can change multiple mappings with a special-purpose scripting language.

The visualization responded to user movements in a number of aspects (see Figure A.7). A colored sphere was drawn at the location of each object being tracked. The color of the spheres changed in response to the objects' average proximity to each other, using warmer colors the greater the distance between the objects. Trailing afterimages were rendered for each sphere, with the afterimage elongating the greater the velocity of that object. Each sphere also regularly emitted particles which flew in the same direction and velocity as the sphere that created them at the point in time when they were spawned. The background included multiple intersecting, rotating wireframe spheres whose rotation speed and color were tied to the average velocity of the tracked objects, while the background itself grew brighter and darker in response to tracked objects' average proximity. Transparency effects were used to blend the various visualization components.

A.3.3. Interactive Sonification Markup Language

In this section, I discuss in detail the Interactive Sonification Markup Language (ISML) [55] which I developed as a special-purpose scripting language in the third phase of the development described in section A.3.2. Interactive sonification is multidisciplinary by nature, including computer science, psychology, HCI, acoustics, music and sound design, etc. My intent was to promote greater efficiency in implementing experimental parameters by demonstrating a method for allowing sonification researchers, even those with no programming experience, to efficiently alter the software specifications of an interactive sonification research platform.

To this end, ISML needed to be flexible enough to support a wide variety of experimental setups, yet simple enough to be easily produced and parsed. Once the initial language specification was complete, my work focused on creating a graphical user interface (GUI) that could facilitate the creation of ISML scripts by non-programmers.

To make generating these scripts easier, ISML includes a web-based GUI so that researchers can create a script simply by responding to prompts, rather than needing to learn ISML's syntax and semantics. This method of configuring the mappings is more efficient and accessible to non-technical researchers than attempting to reprogram the system for every new experiment. Figure A.8 illustrates ISML's role in the iISoP system.

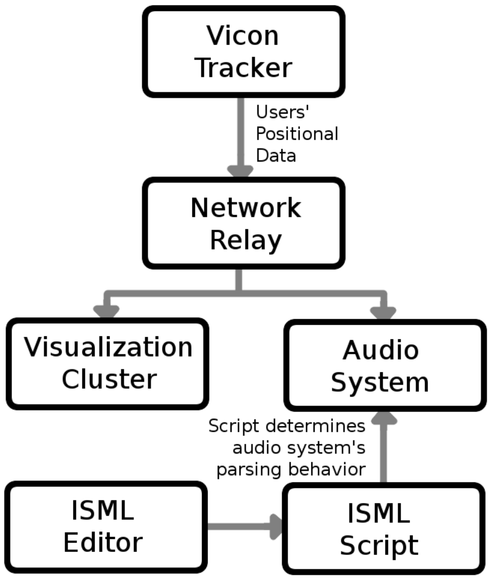


Figure A.8. ISML's role in the iISoP system. The visual component of the network relay can be implemented with DGR.

The format of ISML's syntax is loosely inspired by markup languages, a system for annotating documents so that the annotations are distinguishable from the actual content [7]. Two major markup language standards are Standard Generalized Markup Language (SGML) and Extensible Markup Language (XML), the latter of which is a simplified version of the former designed to maintain its most useful aspects. [64][6] While markup files are typically “documents” of various kinds, they can also represent arbitrary data structures. [17] ISML files, which can be thought of as an activity flowchart or state machine defining application behavior in response to external inputs, are more similar to the latter. Note that ISML is only inspired by these standards and does not make any attempt to conform to them.

ISML also bears similarity to scripting languages, since it is designed to automate the behavior of iISoP's sound generation application. Specifically, it is an example of an audio synthesis scripting language. Other languages of this type exist. A non-exhaustive list of examples includes the ChucK audio programming language [57], the C-based audio programming language Csound [2], the commercial Reaktor software, and the MPEG-4 Structured Audio standard [18]. Some of these systems use graphical interfaces for their scripting languages, while others are textual. Additionally, some of them support live coding, the ability to change the program's behavior while it is running.

By comparison, ISML scripts can be created in a graphical or textural manner. The present version of ISML does not support live coding. To my knowledge, ISML is the only scripting language specifically designed to translate physical movements into dynamically generated sound, which makes it very different from other audio scripting languages.

System Overview The iISoP system dynamically changes sound output via the following procedure.

First, the system checks to see whether a set of *conditions* has been met; for example, "The user is currently moving at a speed equal to or greater than 1 meter per second." If these conditions are met, the system executes one or more *actions*; for example, "Change the key signature to C-major and the time signature to 4/4." Conditions are optional: It is allowed to specify a set of actions that is executed all the time, without any conditions being met.

Each set of conditions and actions is organized into an *activity*. Activities serve as a mechanism for grouping conditions and actions together. By having multiple activities, it is possible to have different sets of conditions and actions which can be checked and executed. Within an activity, all of the conditions must be satisfied in order for the actions to be executed; outside of that activity, its conditions do not matter. Thus, considered collectively, activities and their conditions form a disjunction of conjunctions.

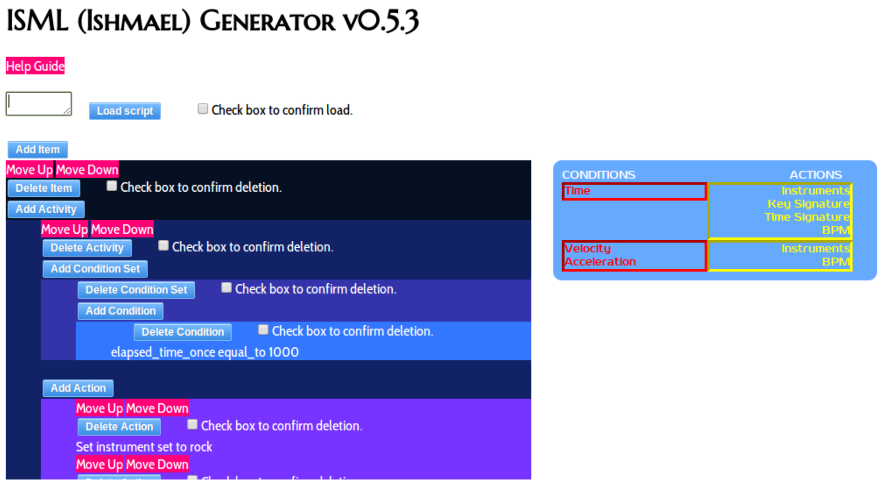
Lastly, activities are organized into *items*. Items provide a scoping mechanism for variables. ISML has 26 variables (a-z) available for use. However, each variable exists independently within the item in which it is used; that is, each variable has item scope. Within each item, all of that item's activities are executed in sequential order from top to bottom. Each item, and all of its activities, are executed (all conditions are checked and the corresponding actions are executed) every cycle of the system. There is no limit on the number of items an ISML script may contain.

The version of ISML I developed allowed for two basic types of conditions: *object* and *comparison*. An object condition applies only to an object in the Vicon tracking system that possesses the user-specified name; for example, “left_foot.” A comparison condition compares whether two values are equal to, greater than, or less than each other. Values that can be compared include constants, variables, current beats per minute, current velocity (in any of the X, Y, or Z axes, or the composite velocity on all three axes), average velocity, acceleration, proximity (the average distance between all tracked objects), current position, and elapsed time (which may be repeatable for conditions that are checked at regular intervals).

Additionally, ISML allows for the following types of actions:

- Assignment. Sets one value equal to an expression. An expression may be another value, or two other values with some arithmetic operation applied on them (e.g., a=b*c).
- Set the key signature.
- Set the time signature.
- Set the instruments being used.
- Play a specific sequence of notes.

A complete language specification for ISML can be found in [55].



Check box to confirm load.
Check box to confirm deletion.
Check box to confirm deletion.
Check box to confirm deletion.
Check box to confirm deletion.
elapsed_time_once equal_to 1000
Check box to confirm deletion.
Set instrument set to rock

Figure A.9. *Screen capture of the ISML graphical editor.*

Graphical Editor Since ISML is designed to be usable by non-programmers, it includes a GUI for the creation of scripts via prompts. The GUI is written entirely in Javascript, so it can be run in a Web browser; and because it does not use any server-side code, it can be run from any computer.

The GUI's appearance is shown in Figure A.9. Upon startup, the user is presented with a blank script and can either access a comprehensive help guide, load an existing script, or begin creating a new script by clicking the “Add Item” button. From here, the GUI dynamically generates additional buttons for adding and deleting the various components of the script under development. To prevent mishaps, selecting a checkbox is required before deleting script components.

To aid user comprehension, each component of the GUI is indented and color-coded so that distinguishing different activities, condition sets, and so on, is more intuitive. Additionally, a “mapping summary” is generated on the side of the screen which summarizes the motion-to-sound mapping that has been created so far. These features help lighten the cognitive burden on the user by keeping the visual representation of the script organized.

Once editing is complete, the user can download the valid ISML-formatted file by clicking the “Download ISML File” button. This file can be further tweaked by hand if desired, or loaded back into the GUI editor at any time for later revision.

A.3.4. High-Tech Art

The iISoP system was used in a collaboration with a world-renowned artist, Tony Orrico, for an artistic performance. [23] ISML was used to tune iISoP’s audio and visual components. The audio was modified to provide soft ambient noise to accompany Orrico's motions, while the visual system was modified to leave permanent lines behind so that it could act as a three-dimensional canvas. Orrico donned a large number of tracking markers and did three performances using the iISoP system (see Figure A.10). This interdisciplinary project exposed Michigan Tech and Finlandia University students to the scientific and technological side of the arts. Orrico got to see and hear his performances in new ways and gained ideas for further exploration. The Tri-M Lab researchers were able to gather large amounts of data from this successful test of the newly built system.

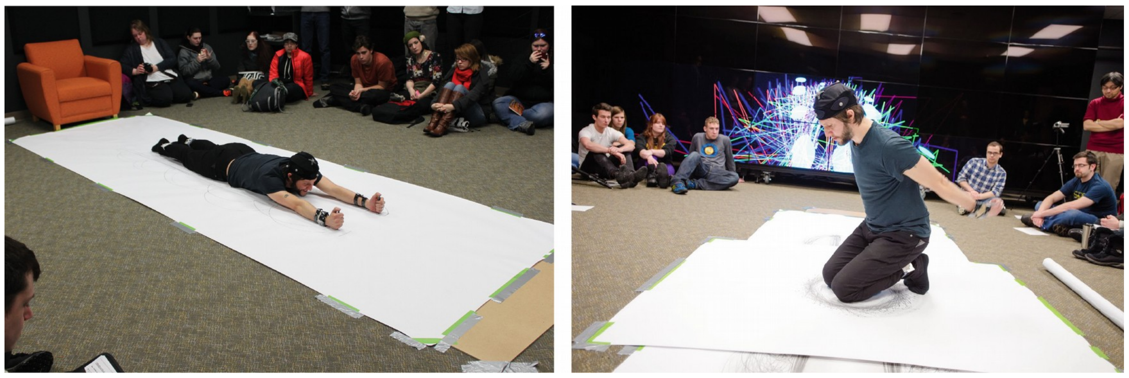


Figure A.10. While Orrico drew on the canvas, his movements also created drawings on the virtual canvas and sonification.

A.4. Robotics Simulation

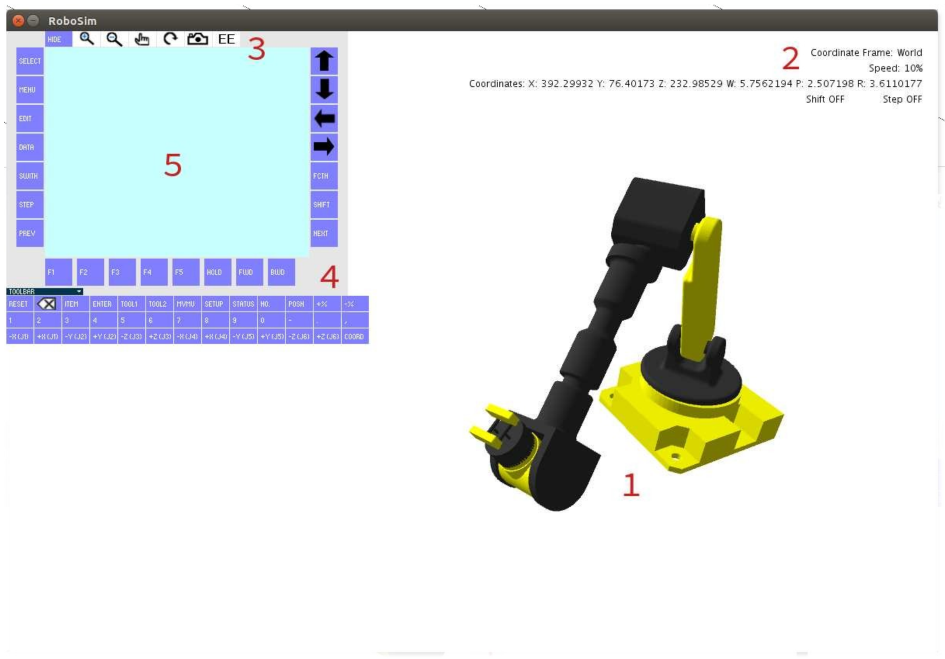
A.4.1. Background

I participated in a collaboration between Michigan Technological University and Bay de Noc Community College to provide education and training in using and programming industrial robots. The objectives of this project were to develop free, open-source industrial robotics training software called RobotRun, and to develop curricula and training materials to supplement the software. [27] I led the development of the first version of RobotRun, which I describe in detail.

A.4.2. The RobotRun Software

I began development on the RobotRun software in the summer of 2015. The software is written in Processing and uses OpenGL to display a real-time 3D visualization of a robotic arm. At the time of my involvement, the software was tested and worked on Windows and Linux.

An overview of the graphical user interface is shown in Figure A.11. The figure shows (1) the robotic arm which was modeled using CAD software and imported into my program with custom software. The arm was designed to resemble the appearance of typical industrial robots. Figure A.11 also shows (2) a small status display which shows the current active coordinate frame (joint or world), speed, joint rotation, and whether the shift and step buttons are on or off. A (3) utility toolbar has buttons for hiding the interface, moving the camera, making a video recording of the scene, and changing the tool/end effector. When the camera movement button is activated, dragging the mouse on the scene will change the camera’s view on the robot. A (4) teach pendant is displayed on the left side of the screen which resembles real-world teach pendants. The teach pendant includes a (5) text display and all of the buttons needed to program the robot and control other functions. Figure 2 shows a close-up view of the teach pendant with a menu being displayed. The user interacts with the robot primarily through this teach pendant to allow users to practice the skills that they need to efficiently program a real-world robot.



```
Coordinate Frame: World
Speed: 10%
Coordinates: X: 392.29932 Y: 76.40173 Z: 232.98529 W: 5.7562194 P: 2.507198 R: 3.6110177
Shift OFF          Step OFF
```

Figure A.11. An overview of RobotRun interface showing a robotic arm with an end effector (right), a teach pendant (upper left), and overlaid status information (upper right). See the text for a description of the individually labeled items in this figure.

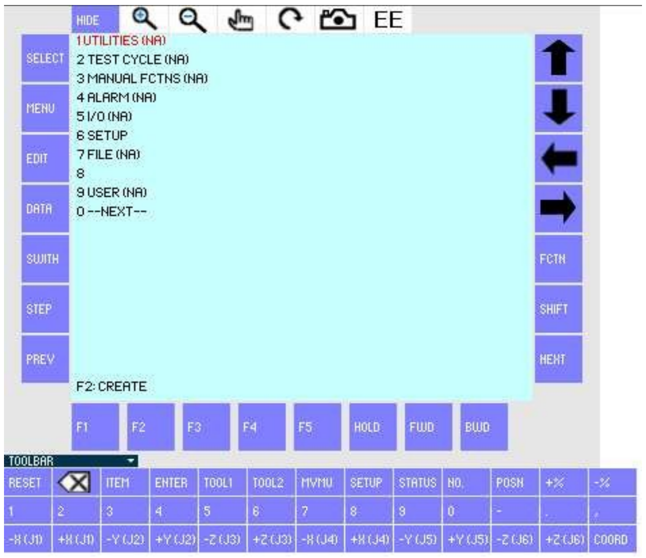


Figure A.12. A closeup view of the teach pendant displaying a menu.

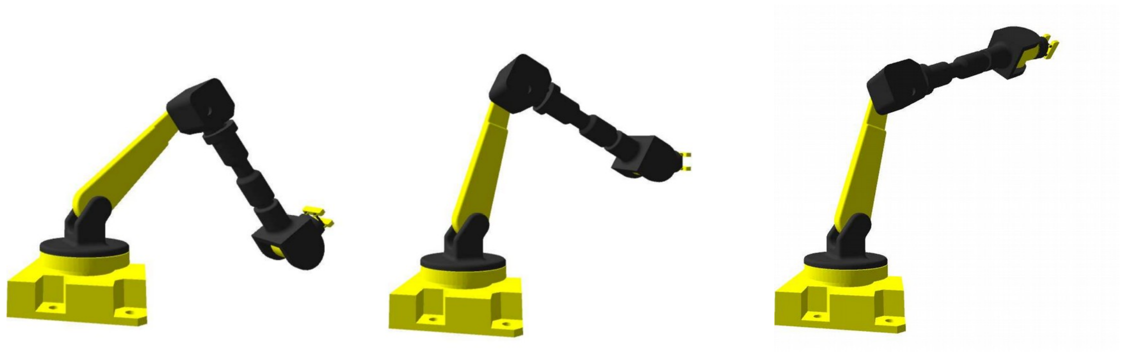


Figure A.13. A series of images showing the robotic arm moving the end effector up in world coordinates using my Inverse Kinematics implementation.

Implementation Details In order to simulate the robotic arm, my software needed an inverse kinematics (IK) solver which enables the software to calculate appropriate joint angles which cause the end effector to be at the appropriate position. The IK algorithm I implemented for the initial version of the software was a variation of Cyclic Coordinate Descent (CCD). For every movement of the robot, a series of points along the path are calculated and then CCD is performed to move the end effector through all of the points on the path. The series of screenshots shown in Figure A.13 illustrate the movement of the end effector vertically in world coordinates.

Each of the robot’s joints can be jogged using the buttons at the bottom of the teach pendant. When one of the buttons is clicked with the mouse, the corresponding joint starts to move until the button is clicked again. There are also buttons to adjust the speed at which the joint changes. When set to operate in the world coordinate frame, jogging the robot changes the position of the end effector and my IK algorithm calculates appropriate joint angles. Initial support was also added to create and save custom frames as an alternative to the world coordinate frame. This is done by having the user select three points that, together with the starting point, form approximately orthogonal line segments, which are then converted into perfectly orthonormal line segments to serve as the axes of a custom coordinate frame. The software internally converts between world coordinates and custom coordinates when performing calculations.

Since this software was intended to be an educational tool, I included a feature which allows users to record videos of their screen along with audio from the user's microphone. This allows a teacher to create videos which demonstrate how to do common tasks or allows a student to record a video of what they have completed and send it to a teacher. This feature is available if fmpeg isinstalled—a free and open-source video/audio encoder and decoder.

Programs for the robot can be created, edited, and run. To create a program, the user enters a name using the keys on the teach pendant. Next, a series of points are recorded by jogging the robot and pressing the appropriate keys on the teach pendant to either record a new point or overwrite an existing point. Motion instructions can be fine (i.e., effectively linear), continuous (the end effector follows a curving path between points), or circular (the end effector traces a path along an arc). The software executes fine motion instructions by linearly interpolating between the current and next point.

For continuous instructions, assuming that the current point is A and the next two points are B and C, the software creates a curved path by linearly interpolating toward a target point that moves from B to point C as the end effector draws nearer to point B. The instruction can specify the amount of curving which controls the speed of the moving target from B to C relative to the movement from A to B. My tests indicate that this approach does as good a job of creating curved paths as, e.g., splines while being less computationally intensive due to the lack of expensive trigonometric function calls.

Circular motion instructions require the user to specify three points, A, B, and C; when executed, the end effector will then trace an arc from A, through B, and to C. The path is calculated by creating a 2D coordinate system from the supplied points, finding the center of a circle whose circumference intersects those three points, calculating intermediate points with the parametric equation

$$P = r * \cos(t) * u + r * \sin(t) * n \times u + \text{center}$$

where t sweeps from 0 to 2π, removing points that do not lie along the desired arc, and finally converting back into the native 3D coordinate frame.

Tool instructions can also be recorded in the program to control the tool on the end effector. I added initial support for different end effectors. The step and shift keys are used when the robot is programmed. They act as toggles where the user presses them once with the mouse to toggle the switch on or of. This simulates holding down the keys—something that can’t be done via the mouse since it is impossible to click and hold multiple buttons onscreen simultaneously. I also completed initial, albeit incomplete support for user-specified frames. Subsequent to my involvement in the project, other researchers continued expanding and improving the software, the current version of which is available for download at http://www.cs.mtu.edu/~kuhl/robotics.

Appendix B: FOW Questionnaire Responses

Participant 1

Did you experience any particular difficulties using the system? Please elaborate.

I found it difficult to maneuver and scroll on the screens. When I tried moving them, they kept overlapping each other.

Is there any aspect of the system you particularly liked? Please elaborate.

I liked the creativity of it.

Is there any aspect of the system you would like to change? Please elaborate.

The 3D aspect.

In what ways was FOW better than using a regular desktop?

There’s a lot more space for windows.

In what ways was FOW worse than using a regular desktop?

Slightly difficult to use.

What other features would you like to see in a virtual reality window manager that are not currently present in FOW?

N/A

Participant 2

Did you experience any particular difficulties using the system? Please elaborate.

Windows could be finicky when you tried to select them. Also, the slider function disrupted the flow of the system since it didn’t always work.

Is there any aspect of the system you particularly liked? Please elaborate.

Being able to use vertical space was nice as far as organizing windows.

Is there any aspect of the system you would like to change? Please elaborate.

When two windows overlap, it could sometimes be hard to reselect the window to move it again—better graphics for showing what window you’re about to select.

In what ways was FOW better than using a regular desktop?

I liked that the keyboard was linked to the text editor—even if I was interacting with a different window, I could type into the text editor.

In what ways was FOW worse than using a regular desktop?

I don’t think I could use it for longer than 30-60 minutes. I began to just want to get the activity done with towards the end when I started experiencing eye strain.

What other features would you like to see in a virtual reality window manager that are not currently present in FOW?

Designed for 180 degrees rather than 360 [would be] more comfortable. I wanted to have a second layer where I could stash windows when I didn’t want to be looking at them.

Participant 3

Did you experience any particular difficulties using the system? Please elaborate.

I found it difficult to choose new windows to scroll in them. Also it was hard to put windows where I wanted them.

Is there any aspect of the system you particularly liked? Please elaborate.

I liked being able to use horizontal movement in order to find data to type in the text box, as it was more convenient than switching away from a window.

Is there any aspect of the system you would like to change? Please elaborate.

The only aspect I would like to change is the ability to scroll in new windows even if I have another one that was previously selected.

In what ways was FOW better than using a regular desktop?

It was easier to enter in data and better than switching between windows.

In what ways was FOW worse than using a regular desktop?

The resolution made it hard to see the words sometimes, and it was hard to focus. I also sometimes lost track of my finger position, making it hard to type.

What other features would you like to see in a virtual reality window manager that are not currently present in FOW?

The only thing I can think of is the ability to see your hands if you move away from the keyboard, but I don’t know how that would be possible.

Participant 4

Did you experience any particular difficulties using the system? Please elaborate.

Without being able to see the keyboard, I had some difficulty finding my home keys. The cursor was quite sensitive and did not always release windows when I wanted to. In addition, all but the window directly in front of me was quite out of focus.

Is there any aspect of the system you particularly liked? Please elaborate.

Being able to change windows at a glance was incredibly convenient.

Is there any aspect of the system you would like to change? Please elaborate.

I would prefer a “drag and drop” system when ti comes to moving windows around the screen, as toggling drag on and off can be somewhat counterintuitive. I would also recommend a more textured keyboard, to make it easier to find home keys.

In what ways was FOW better than using a regular desktop?

Viewing windows at a glance was easier than selecting them individually.

In what ways was FOW worse than using a regular desktop?

Constantly turning my head while wearing the headset caused some amount of neck strain. Not being able to see the keyboard severely affected my typing.

What other features would you like to see in a virtual reality window manager that are not currently present in FOW?

Easier scrolling within windows, being able to minimize windows not in use, and some kind of finger placement indicator would enhance the experience.

Participant 5

Did you experience any particular difficulties using the system? Please elaborate.

The only difficulty I had in the beginning was that I couldn’t locate the cursor, but later I was able to do so.

Is there any aspect of the system you particularly liked? Please elaborate.

I liked how I could keep all the windows stacked up according to my comfort.

Is there any aspect of the system you would like to change? Please elaborate.

I would like to make the colors and brightness more soothing for eyes.

In what ways was FOW better than using a regular desktop?

I could open a lot more windows at the same time and focus on the one I need at that moment by just rotating my head.

In what ways was FOW worse than using a regular desktop?

It would be very difficult ot use the FOW for a long period of time as compared to a regular desktop.

What other features would you like to see in a virtual reality window manager that are not currently present in FOW?

The selecting part was a bit loosely designed. There were times when I was trying to unselect a window, but that wasn’t happening because the mouse cursor wasn’t in the center at t that time. Apart from that, as I mentioned above, color and brightness options should be there so that every user can customize them according to his requirement.

Participant 6

Did you experience any particular difficulties using the system? Please elaborate.

Moving the cursor within the text box was difficult. My contacts dried out frequently. Was difficult to read text at the fringes.

Is there any aspect of the system you particularly liked? Please elaborate.

It was fun! Was easy to move boxes & the keyboard helped to type.

Is there any aspect of the system you would like to change? Please elaborate.

Textbox could be easier to manage.

In what ways was FOW better than using a regular desktop?

Easier to overlay windows.

In what ways was FOW worse than using a regular desktop?

Wasn’t able to see as much in one place of vision—smaller windows than a desktop.

What other features would you like to see in a virtual reality window manager that are not currently present in FOW?

Not sure, I’m not knowledgeable about what’s possible.

Participant 7

Did you experience any particular difficulties using the system? Please elaborate.

Window focus being tied to the grabbed window rather than the last clicked window was awkward.

Is there any aspect of the system you particularly liked? Please elaborate.

The large amount of space.

Is there any aspect of the system you would like to change? Please elaborate.

Window focus, as mentioned above. Also, some way to rotate the center reference would be nice. Additionally, the area seemed a little more cramped than necessary.

In what ways was FOW better than using a regular desktop?

The extra space is useful for many applications. This eliminates the need for remembering many hotkeys.

In what ways was FOW worse than using a regular desktop?

Generally, polish. Scrolling, focus, drag-and-drop and windows resizing. Nothing that couldn’t be easily fixed.

What other features would you like to see in a virtual reality window manager that are not currently present in FOW?

See above for the basics, but I would like automatic window placement, like in tiling window managers.

-Workspaces
-Rotate center reference

Participant 8

Did you experience any particular difficulties using the system? Please elaborate.

Windows at the edges were blurry. So while typing (keeping this window at the center) it was difficult to read the windows at the left/right. So had to adjust.

Is there any aspect of the system you particularly liked? Please elaborate.

The virtual reality!!
The virtual keyboard—liked & disliked
The fact that I could place my windows in whichever direction I like.

Is there any aspect of the system you would like to change? Please elaborate.

The virtual keyboard should be placed a little more at the bottom to avoid blocking the center windows. The scroll bar is difficult to use.

In what ways was FOW better than using a regular desktop?

Spacious, flexible mobility.

In what ways was FOW worse than using a regular desktop?

Habituated in using desktop and physical keyboard. Virtual keyboard was interesting but difficult to find the keys, so task became slower as I misspelled or kept searching [for] a key. Couldn't use shortcut keys. Too much mobility of windows.

What other features would you like to see in a virtual reality window manager that are not currently present in FOW?

Shortcuts, flexible scroll button.
Clear image in all 360 degree angles.
Viewing left-right, top-bottom was difficult due to blurriness.

Participant 9

Did you experience any particular difficulties using the system? Please elaborate.

To locate the keys.

Is there any aspect of the system you particularly liked? Please elaborate.

Yes, it is interactive.

Is there any aspect of the system you would like to change? Please elaborate.

Instead of escape, using some buttons to close.

In what ways was FOW better than using a regular desktop?

360 degreeview of windows.

In what ways was FOW worse than using a regular desktop?

To locate keys and switching between windows.

What other features would you like to see in a virtual reality window manager that are not currently present in FOW?

Participant 10

Did you experience any particular difficulties using the system? Please elaborate.

The disconnect between the keyboard and mouse focus made editing text difficult. Scrolling using the scrollbar seemed impossible. Mouse seemed to move too fast, perhaps due to the small projection sphere. Screen would detect movement even when head was stationary.

Is there any aspect of the system you particularly liked? Please elaborate.

I liked the ability to move windows, and scrolling with the keyboard was necessary. The virtual keyboard helped reduce errors once I found the keyboard, but a visual indicator of the keyboard and mouse position would be more useful.

Is there any aspect of the system you would like to change? Please elaborate.

It would be nice to rotate the view, to re-anchor the view. This would reduce the need to rearrange all the windows, given that the keyboard and mouse are typically stationary.

In what ways was FOW better than using a regular desktop?

Being able to place windows arbitrarily was useful. I could put less used [windows] slightly farther away, and intuitively turn to see them when necessary.

In what ways was FOW worse than using a regular desktop?

The windows were not resizable, which necessitated scrolling. The scrolling behavior was not intuitive. The depth order of the windows didn't make it easier to overlay windows. Text was difficult to focus on, ended up seeing pixels instead.

What other features would you like to see in a virtual reality window manager that are not currently present in FOW?

I think having present locations for windows could make it easier to manage different appplications. Zoom in/out. Depth order/alt-tab behavior. Ability to focus with mouse hover and/or couple keyboard and mouse focus. Some hot-key or indicator to point to the current application that has focus.

Participant 11

Did you experience any particular difficulties using the system? Please elaborate.

The inability to resize the windows.

Is there any aspect of the system you particularly liked? Please elaborate.

I liked being able to have big windows in view from left to right.

Is there any aspect of the system you would like to change? Please elaborate.

The resolution of the HMD.

In what ways was FOW better than using a regular desktop?

More space, more ability to focus because it eliminated surrounding distractions, and windows are closer to my face.

In what ways was FOW worse than using a regular desktop?

Some of the disable functions/keys.

What other features would you like to see in a virtual reality window manager that are not currently present in FOW?

All of the things listed above.

Participant 12

Did you experience any particular difficulties using the system? Please elaborate.

Edge of the display blurry, corrected by moving window to center of display.

Is there any aspect of the system you particularly liked? Please elaborate.

360-degree work environment saves space.

Is there any aspect of the system you would like to change? Please elaborate.

Ability to use mouse scrolling instead of keyboard arrows.

In what ways was FOW better than using a regular desktop?

Multiple windows available seems more convenient than using tabs.

In what ways was FOW worse than using a regular desktop?

Loss of sight for hands on keyboard.

What other features would you like to see in a virtual reality window manager that are not currently present in FOW?

Way to locate an off-screen mouse cursor.

Participant 13

Did you experience any particular difficulties using the system? Please elaborate.

A bit hard to move the sliders using the mouse.

Is there any aspect of the system you particularly liked? Please elaborate.

1. It is good to have the window you want to use in one compact space. It is very easy to find them.
2. The virtual keyboard is very useful to locate the keys that you are not familiar with.

Is there any aspect of the system you would like to change? Please elaborate.

The slider.

In what ways was FOW better than using a regular desktop?

A lot more windows can be present at the same time.

In what ways was FOW worse than using a regular desktop?

The display can be blurry on the sides.

What other features would you like to see in a virtual reality window manager that are not currently present in FOW?

Maybe using gesture or simply using the arrow keys to switch the placement of two windows.