Object Sliding and Beyond: Investigating Object Manipulation in 3D User Interfaces

by

Junwei Sun

M.Sc., Western University, 2012

B.Sc., Peking University, 2011

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

in the
School of Interactive Arts & Technology
Faculty of Communication, Art and Technology

SIMON FRASER UNIVERSITY

Fall 2019

Approval

Name:	**Junwei Sun**
Degree:	**Doctor of Philosophy**
Title:	**Object Sliding and Beyond: Investigating Object Manipulation in 3D User Interfaces**
Examining Committee:	**Chair:** Cheryl Geisler Professor
	Wolfgang Stuerzlinger Senior Supervisor Professor
	Bernhard Riecke Supervisor Associate Professor
	Chris Shaw Supervisor Professor
	Lyn Bartram Examiner Professor
	Kyle Johnsen External Examiner Associate Professor School of Electrical and Computer Engineering University of Georgia
Date Defended/Approved:	September 23, 2019

Ethics Statement

The author, whose name appears on the title page of this work, has obtained, for the research described in this work, either:

a. human research ethics approval from the Simon Fraser University Office of Research Ethics

or

b. advance approval of the animal care protocol from the University Animal Care Committee of Simon Fraser University

or has conducted the research

c. as a co-investigator, collaborator, or research assistant in a research project approved in advance.

A copy of the approval letter has been filed with the Theses Office of the University Library at the time of submission of this thesis or project.

The original application for approval and letter of approval are filed with the relevant offices. Inquiries may be directed to those authorities.

Simon Fraser University Library
Burnaby, British Columbia, Canada

Update Spring 2016

Abstract

3D manipulation is one of the fundamental tasks for interaction in virtual environments. Yet, it can be difficult for users to understand the spatial relationships between 3D objects and how to manipulate them in a 3D scene, as, unlike in the physical world, users do not have the same visual cues for understanding scene structure or can leverage constraints and affordances for interaction. My goal is to create better user interface for 3D manipulation platforms, with a focus on positioning objects. I designed efficient, accurate, and easy-to-use 3D positioning techniques for both desktop and virtual reality (VR) systems. My work also contributes guidelines for designing and developing 3D modelling software for desktop and VR systems, and enable 3D content designers, game designers, or even novice users to benefit from improved efficiency and accuracy for 3D positioning tasks.

Much of my thesis work builds on a 3D object sliding technique, where objects slide on surfaces behind them, which helps with some positioning tasks. First, I improved 3D positioning on a desktop system, with the mouse and keyboard as input devices. I presented two new techniques that significantly outperform the industry-standard widget-based 3D positioning technique for tasks involving floating objects or objects that can be at multiple positions in visual depth. Second, I proposed a new technique that allows users to select and position hidden objects. The new technique also outperformed 3D widgets. Then, I applied my techniques in a VR system with a head-mounted display (HMD) and compared the performance of different input devices. I found that the combination of the mouse with my new positioning technique is still the best solution, even in VR. In the remainder of my thesis work, and focusing on tasks involving more distant objects, I investigated manipulation techniques in VR that do not rely on the availability of a mouse. I designed and implemented a technique that significantly improved the accuracy for 3D positioning tasks for targets that were in contact with the scene.

Keywords: 3D positioning; object sliding; input devices; 3D selection

This thesis is dedicated to my parents,

for their unconditional love, support, encouragement,

and being so awesome.

Acknowledgements

I would like to thank my senior supervisor Dr. Wolfgang Stuerzlinger for always supporting me during my Ph.D. study, for his patience, enthusiasm, and kindness. He encouraged me to explore different directions within my research area and assisted me much when I encountered difficulties. He showed me how to become a better researcher and a better person. I feel privileged to have worked with him. This thesis would not have been possible without him.

I also thank my supervisors Dr. Bernhard Riecke and Dr. Chris Shaw for providing new perspectives on my research, especially their valuable suggestions on my thesis proposal. And I thank Bernhard for teaching me quantitative research methods.

I want to give special thanks to all my lab mates for always motivating me. Their friendship is priceless. It has been my honor to have them alongside me throughout this long journey.

To all my colleges and friends, thank you for the help and companionship from work to daily life.

To my parents, my grandma, and my entire family, thank you for your love and support regardless of the distance between us. I miss you deeply.

I would like to thank all my participants of my user studies for their patience and valuable feedback.

Table of Contents

List of Tables

List of Figures

List of Acronyms

3DUI	3D User Interface
CAD	Computer-Aided Design
DOF	Degree of Freedom
GPU	Graphics Processing Unit
HMD	Head-Mounted Display
VR	Virtual Reality

Chapter 1.

Introduction

A 3D user interface (3DUI) is a user interface where 3D interaction takes place, i.e., the user's tasks occur directly within a real or virtual 3D space. 3D user interfaces primarily allow users to interact with virtual objects, environments, or information. Examples of 3D user interfaces include gaming systems, modelling applications, virtual and augmented reality systems, scientific visualization, etc. A 3D user interface is more difficult to design, implement, and use than a 2D user interface. The challenges for creating good 3DUIs can be attributed to diverse sources: users' perceptual, cognitive, and motor abilities, limitations of available input and output devices, the more complex nature of 3D tasks relative to 2D ones, and the large variety of implementation strategies and development environments [Herndon et al., 1994].

3D manipulation is one of the fundamental 3DUI tasks for interaction in both virtual and real environments. The quality of the manipulation techniques that allow us to interact with 3D virtual objects has a profound effect on the quality of the 3D user interface. In fact, a defining feature of virtual reality (VR) is the ability to manipulate virtual objects interactively, rather than simply viewing a passive environment [Bowman et al., 1997]. If users cannot manipulate objects effectively in virtual environments, many application-specific tasks simply cannot be performed [Bowman et al., 2004]. The field of 3D manipulation research has been growing rapidly in the last few decades. Nowadays, because of the increasing availability of virtual reality hardware and software, there is a growing demand for novel and efficient manipulation techniques. The design of 3D manipulation techniques is thus an important challenge.

In the real world, 3D manipulation refers to the act of handling 3D objects with one or two hands. In 3D virtual environments, users often encounter the need to arrange objects in a scene. There are many variations of manipulations tasks characterized by application goals, object sizes, where the objects are in the scene and where their target positions are, especially if they are further away from the user, the physical and

psychological states of the user, etc. The effectiveness of 3D manipulation techniques greatly depends on the tasks. However, despite this variety, three tasks are considered to encompass basic manipulation: selection, positioning, and rotation.

Selection is the task of acquiring or identifying one or more objects to subsequently manipulate from the 3D environment. Selection is the initial task for most common user interactions in a virtual environment. The real-world counterpart of selection is picking up an object with one or two hands.

Positioning refers to the task of changing the 3D position of an object, where 3 DOFs have to be controlled. Depending on the display platforms and the choice of input devices, different mappings have to be used for 3D positioning tasks. The real-world counterpart to positioning is moving an object from a starting position to a target position without changing its orientation.

Rotation is the task of changing the orientation of the object, usually around the object's center, where 3 DOFs have to be controlled as well. The real-world counterpart of rotation is rotating an object from a starting orientation to a target orientation.

In my thesis, I focus on 3D selection and positioning, i.e., selecting an object and then changing its position. Moving objects is a basic, yet frequent task in 3D user interfaces. For 3D positioning, I concentrate on speed, precision, and usability. Although fast and accurate 3D object positioning is not required in some applications, such as casual games, speed and precision are extremely important for other applications, such as creating 3D models for interior and exterior design, industrial design, 3D animations, architectural mockups, or homebuilder applications. Other applications include 3D games that involve moving virtual objects under time pressure, including some 3D puzzle games. I thus investigate easy-to-use techniques that allow users to select an object easily and position it with speed and precision.

Besides tasks, 3D manipulation can be classified through different characteristics, such as platforms, input devices, DOF of technique, etc. Platforms for 3D manipulation include desktops, mobile devices, VR HMDs, immersive CAVEs, augmented reality headsets, etc. Input devices for 3D manipulation include mouse and keyboard, touchscreens, user's hand, stylus, VR style controllers, etc. The number of supported DOFs is one of the most important characteristic of input devices. The mouse,

one of the most commonly used input devices, has 2 DOFs. Some devices allow 3 or 6 DOFs as input.

3D manipulation techniques can also be classified by how many DOFs they can control simultaneously. Some techniques allow users to control 2 DOFs simultaneously, while others permit controlling 3 DOFs or even more. For any specific manipulation task, there are some input devices that are more appropriate than others. Device ergonomics, the number and type of input modes, and the types of tasks all play a role in choosing a suitable input device. There is not a single best input device or technique that is universally beneficial for all 3D manipulation tasks.

Typically, a 2DOF input device is better suited for a 2DOF technique, and a 3DOF input device is better suited for a 3DOF technique. For 3D manipulation tasks, using a 2DOF input device could introduce a 2D/3D mismatch. However, this can be addressed with smart mappings between user input and object movement [Bowman et al., 2004]. For example, the Arcball technique used a 2DOF mouse to perform 3DOF rotation [Shoemake et al., 1992]. The triangle cursor [Strothoff et al., 2011] used a 2DOF touchscreen as input device, yet users were able to control 4 DOFs simultaneously (3 for translation and 1 for rotation) through multiple simultaneous touches.

In my thesis, I explore different combinations of platform, input device, and technique for 3D manipulation tasks, with a focus on precise 3D object positioning. I aim to fill in a gap in the current literature, see research questions below. Besides performance measures such as speed and precision, I also prioritize the usability of the 3D positioning techniques. I aspire to find combinations of input device and technique that are easy to learn and use, while still being fast and accurate.

In the real world, objects usually stay in contact with other objects. Floating objects are rare and are usually only temporarily not suspended. Based on the observation of real-world interactions, the contact-plane sliding technique [Oh et al., 2005] linked mouse movement directly to object movement and moves an object on the surface behind it, i.e., uses the contact of the object with that surface. The contact-plane sliding technique makes three fundamental assumptions: A) The manipulated object stays by default in contact with the rest of the scene, B) The manipulated object does not interpenetrate the scene, and C) The manipulated object is always at least partially

visible to the user. These three assumptions (contact, non-collision, visible) are true for almost all scenes in the real world. Much of my thesis work is inspired by the contact-plane sliding technique, yet I introduce significant extensions and improvements for this technique to accommodate different tasks on different platforms.

In Section 1.1, I present the research questions I am addressing. I also discuss the motivations behind these questions. The five research questions will be addressed in detail in Chapter 2 to 5. In Section 1.2, I list my contributions to the field of 3D manipulation. I outline my thesis structure in Section 1.3.

1.1. Research Questions and Motivations

Here, I list the research questions I am addressing and the motivations behind them.

RQ1: How much does extended sliding improve the speed/accuracy of 3D positioning in a desktop system?

The mouse is the most commonly used input device in desktop systems. The physical form of the mouse ensures that the user is not restricted to any particular grip, which reduces fatigue. The mouse is easy to acquire and physically stable due to the friction a desktop surface affords. Moreover, the mouse translates 2D hand movements directly into 2D cursor movements on the screen. Even though the mouse is spatially disassociated from the screen, this movement, in combination with the apparent immediate and isomorphic effect on the screen, supports the users' perception that they are moving the cursor on the screen themselves. Finally, people are extremely familiar with the form and function of the mouse.

Since the mouse is commonly used and people are very familiar with it, my goal is to improve 3DOF manipulation with the mouse. Mouse-based 3D manipulation can only control 2 DOFs at a time, e.g., simultaneous translations along all three directions are not possible. This can be compensated by proper mappings and the use of constraints. The mouse has been proven to be a reliable input device for 3D manipulation, despite the lack of the ability to directly control a third DOF (or more) [Bérard et al., 2009].

Computer-aided design (CAD) enable users to create, modify, or optimize a design of 3D objects and scenes. CAD is commonly used for detailed engineering of 3D models. When using CAD software, users often encounter the situation where they must move virtual 3D components around in the scene. Speed and precision are often required in the task of moving 3D objects. This is usually achieved by the usage of 3D widgets [Conner et al., 1992]. Before moving a 3D object, the users must identify it. In a desktop CAD software, this is usually accomplished by clicking on the object with the mouse button, which selects it. This is a form of image plane selection [Pierce et al., 1997].

Despite its prevalent use in 3D CAD systems, the 3D widgets solution has its limitations. Users must mentally separate any desired 3D movement into multiple 1D or 2D movements. This increase the cognitive burden to users, which can make the task of 3D positioning tedious to perform [Oh et al., 2005]. Also, the widgets could be difficult to select depending on the position and orientation of the object. Also, the widgets usually need to be rendered on top of everything, otherwise they could be occluded by the rest of the scene, other widgets, or the object itself. This could be perceptually inaccurate. Finally, the need to repeatedly select widgets can become time-consuming.

My motivation is to improve 3D positioning in a desktop system using the mouse as input device. I want to design 3D positioning techniques that are easy to use and outperform the 3D widgets solution, the most commonly used technique in industry 3D CAD software. This will benefit 3D content designers or even novice users to achieve better speed and precision in their 3D modelling practice. I would also want to make it possible to include the techniques in standard 3D CAD software.

The above-introduced sliding technique is a contact-based 3D positioning technique that outperforms 3D widgets [Oh et al., 2005]. However, contact-plane sliding has its limitations, as objects can only be placed in positions that meet three underlying assumptions. Thus, I present two techniques that extend contact-plane sliding by breaking the first two assumptions for contact-plane sliding, i.e., allow the user to force the manipulated object to float or collide with the rest of the scene. I rely on the image plane selection method to select the object with a mouse click. I hypothesize that extended sliding will improve 3D positioning in a desktop system. I discuss RQ1 in more detail in Chapter 2.

RQ2: How much does layer-based transparency improve the speed/accuracy of selection and positioning of hidden objects?

In 3D modelling systems, object selection is typically limited to (at least partially) visible objects. When an object is occluded, i.e., visually blocked by other objects, it is then impossible to select it in a single perspective view. Hidden objects are also difficult to manipulate precisely, as no visual cues are provided to the users. Some research focussed on selecting targets that are completely occluded from the user's viewpoint [Wunsch et al., 1997]. One simple solution is to navigate to a new viewpoint so that the desired object is not occluded. Yet, for selection-intensive applications, navigating to a viewpoint that permits (easier) selection can be time-consuming, and requires navigation effort. Camera navigation can also be confusing and time-consuming for novice users. Some CAD software uses wireframe rendering to avoid occlusion completely. Pursuing a different approach, Elmqvist et al. [Elmqvist et al., 2008] reviewed a broad range of occlusion management techniques. However, there is still no optimal solution that solves every occlusion issue for 3D selection.

My motivation is to further improve 3D selection in a desktop system with the mouse as input device. I want to design a selection technique that can select hidden objects without changing the viewpoint. Ideally, I also want to combine the selection technique and object sliding directly with a seamless transition. Users would be able to start object sliding right after selection without having to release the mouse button. This will make it faster to perform positioning tasks. Users would then be able to select an object from an initial fully hidden position and move it to a fully hidden target position, even in complex scenes. I would also want the techniques to be compatible with the interaction methods used in standard 3D CAD software.

In the above discussion of RQ1, I presented two techniques for 3D positioning in a desktop system. In both techniques, I used image plane selection, i.e., click on the object visible in a perspective view with the mouse button. However, these extended sliding techniques are not able to deal with scenes where objects are completely hidden. To address this issue, I designed and implemented a layer-based technique that makes specific layers of the scene semi-transparent, obviating the need to move the camera. As perspective and occlusion are the strongest cues for judging 3D position [Wickens et al., 1999], revealing the perspective scene layer by layer increases the number of depth

cues available for object selection (and later manipulation). I also applied a transparency mask on the scene layers during object sliding, making the object always fully visible during manipulation, even if the user moves the object behind others. I hypothesized that users will be able to select and slide hidden objects quickly and accurately with transparency sliding. Also, I hypothesized that transparency sliding will outperform 3D widgets in 3D positioning tasks. I discuss RQ2 in more detail in Chapter 3.

RQ3: How do input devices compare for object sliding in VR?

Virtual reality (VR) headsets have become prevalent in recent years and are starting to be widely used with applications of entertainment as well as productivity. There has been research of integrating CAD and virtual environments [Whyte et al., 2000], and various 3D CAD applications for VR have been presented. These applications enable users to design and build worlds in a virtual reality environment, which has the potential to inspire 3D designers in ways that desktop systems cannot, as the designer can then be “inside” the 3D world they are creating. Typically, these applications use a 6DOF controller as the input device.

Input devices allow users to communicate with the interface, which makes them an important component in designing, developing, and using 3D user interfaces. There are many different types of input devices to choose from, and some of them may be more appropriate for certain tasks than others. Despite the dominance of the mouse in 2D user interfaces, there is still not a single best device suitable for all tasks in 3D user interfaces. As mentioned in the discussion of RQ1, the mouse is a great input device for 3D tasks in desktop systems. However, the mouse is typically not designed to be used in immersive 3D environments that involve HMDs, since it needs to be placed on a 2D surface to function properly.

Although precise 3D positioning is not always necessary in entertainment applications such as VR games, it is still an important task for applications such as 3D modelling in VR. Even though a 3DOF tracked controller is the most commonly used input device in VR environment with HMDs, I am still interested in evaluating the performance of the mouse for precise 3D positioning in such environments. My motivation is to find the best input device for precise 3D positioning in VR, with the aim of providing guidelines for designing and developing 3D CAD applications in VR. Since

sliding outperformed 3D widgets on the desktop, I choose sliding as the positioning technique for this research question.

Although the mouse is an ideal input device for 2D interfaces and some 3D interfaces [Zhai, 1998] [Bérard et al., 2009], there has been little research that evaluated the suitability of the mouse for 3D positioning tasks in HMDs. Extended sliding performs well on the desktop but has not been evaluated in a VR system. Thus, I implemented contact-plane sliding in VR and compare the performance of different input devices with this system. I hypothesize that the mouse would outperform the controller and the touchpad, due to the 2DOF nature of sliding and users' familiarity of the mouse. I discuss RQ3 in more detail in Chapter 4.

RQ4: Which cursor display performs better for object sliding in VR?

When working with stereoscopic content, the shape and behavior of a standard 2D cursor is not always suitable, as depth conflicts between the cursor and the content can lead to confusion [Argelaguet et al., 2009]. Since the 2D cursor is usually drawn on top of the other objects, it will occlude all the objects that overlap it on the screen projection. This results in a depth cue conflict between interposition and stereopsis: objects in front of the screen will exhibit a negative parallax value, whereas they might appear partially occluded by the cursor which is perceived to be on the screen plane.

Using an appropriate representation of the cursor significantly affects users' perception of the scene during selection. A one-eyed (mono) cursor eliminates stereo cue conflicts by displaying the cursor only to the dominant eye [Ware et al., 1997]. However, the lack of a cursor in both eyes may cause some discomfort with long-term use [Hill et al., 2008]. Still, Teather et al. found that one-eyed cursors improve screen-plane pointing techniques, but also that they are not universally beneficial [Teather et al., 2013].

Although the one-eyed cursor was shown to be beneficial for 3D selection [Teather et al., 2013], it remains to be seen whether it performs well for 3D positioning. My motivation is to find the best cursor display method for object sliding in VR, therefore providing some guidelines for designing and developing 3D CAD applications in VR. I implemented contact-plane sliding in VR and compare the performance of different cursor displays. I hypothesize that the stereo cursor would be perform better, due to the

potential eye fatigue introduced by the one-eyed cursor. I discuss RQ4 in more detail in Chapter 4.

RQ5: How much does extended sliding improve the speed/accuracy of 3D positioning in VR?

Precise 3D positioning is an important task for applications such as 3D modelling in VR. A 3DOF/6DOF controller is the most commonly used input device in immersive environments with HMDs. For mid-air manipulation, 3DOF/6DOF input devices provide more natural input mappings, especially for VR systems that use HMDs. Interacting in virtual environments with more natural actions opens more possibilities for exploiting the richness of the interaction, allowing users to control more DOFs simultaneously and extend well-known real-world actions. However, 3D interaction in virtual environments can be a difficult and sometimes physically demanding endeavor. The design of 3D interaction techniques for virtual environments is thus a challenging problem.

My motivation is to design a technique that improves 3D positioning in VR, using a 3DOF controller as input device. I implemented contact-plane sliding in VR when addressing RQ3 and RQ4. However as discussed above, contact-plane sliding has its limitations, i.e., objects can only be placed to positions that are in contact with the rest of the scene. To address this limitation, I decide to apply the extended sliding techniques in a VR system. For this, I identify appropriate mappings for all the user actions needed for extended sliding. This allows users to position objects at all the visible target positions in the virtual environment. If a 3D modelling application in VR can only be used with a 3DOF controller, my work could improve the precision of 3D positioning.

I performed a comparative study involving extended sliding and full 3DOF positioning techniques. I hypothesize that sliding will not be the fastest technique for 3D positioning in VR. In contact-plane sliding, the manipulated object is constrained to 2DOF movements on a surface. In extended sliding, moving the object in the third dimension, i.e., making the object float or collide, is relatively more time-consuming as the user has to first perform a 2D sliding movement followed by another interaction to position the object in the third dimension. With a 3DOF input device, any mapping for such extended sliding would be less natural than direct 3DOF manipulation. Therefore, I

hypothesize that a 3DOF positioning technique will be faster than extended sliding in VR.

However, I also hypothesize that extended sliding will be as accurate as a full 3DOF technique. Since it is context-based, the extended sliding technique will be more accurate for objects that are in contact or close to a surface in the scene. Also, since extended sliding with the controller is based on ray-casting, I hypothesize that extended sliding will be less fatiguing. I discuss RQ5 in more detail in Chapter 5.

1.2. Contributions

In this section, I list my contributions to 3D manipulation, with a focus on 3D positioning, i.e., moving the selected object to the target position. I explore 3D positioning tasks both on a desktop system and in a VR system. My work will enable 3D content designers, game designers, or even novice users to benefit from the improved speed and accuracy of 3D positioning tasks. They will also provide useful guidelines for designing and developing 3D modelling software for both desktop and VR systems.

Much of my work is inspired by contact-plane sliding, a 2DOF technique designed for 3D positioning on a desktop [Oh et al., 2005]. With this technique, users can easily drag and drop the object in the 3D scene with mouse movement. Most users found this easy to understand and learn. I extended contact-plane sliding to permit floating and colliding objects, while keeping the user interface simple. I also made it easier to move objects in depth. The techniques are still easy to learn and use, as only a single modifier key is required. This allows users to position 3D objects to any visible 3D location with speed and precision. The extended sliding techniques outperforms the industry standard 3D widgets solution [Conner et al., 1992], which motivates me to recommend its inclusion into 3D modelling software. This part of contribution was published in [Sun et al., 2016], and will be discussed in more detail in Chapter 2.

I presented transparency sliding techniques which enable the selection and sliding of entirely hidden objects. The selection technique uses a single modifier key, and no extra interaction is required for the subsequent sliding action. The transparency sliding techniques outperforms 3D widgets, allowing users to select initially hidden

objects and move them to hidden target positions with ease. This part of contribution was published in [Sun et al., 2019], and will be discussed in more detail in Chapter 3.

I started exploring 3D positioning in VR systems with HMDs by applying contact-plane sliding to VR. I compared the performance of different input devices and cursor display methods. The mouse performed the best for object sliding, which serves as a benchmark for 3D modelling applications in VR. If possible, 3D CAD applications should use a VR system that can be used in a seated position with a mouse. Also, I identify that 3D CAD software in VR should use a stereo cursor to generate less eye fatigue. This part of contribution was published in [Sun et al., 2018], and will be discussed in more detail in Chapter 4.

In situations where a mouse is not available in a VR system, a 3DOF controller is the most common input device. To support contact-plane sliding with the contact constraint, I ported my extended sliding technique to VR, using the controller as input device. I identified the appropriate mappings for extended sliding, using a single button for the necessary mode change. I used a mapping that puts the confirmation button on the non-dominant hand to reduce the Heisenberg effect, where the slight hand movement caused by a button press in the confirmation phase can affect the precision of object selection or placement [Bowman et al., 2001]. Users were quickly able to learn the interaction technique in a training session. The extended sliding technique significantly improved the precision of 3D positioning for objects in contact with the scene. After prolonged use of VR applications with the controller mid-air, user fatigue becomes a potential concern. Yet, I found that my technique did not require much hand movement in depth, and thus generated less fatigue. This part of contribution was accepted in [Sun et al., 2019], and will be discussed in more detail in Chapter 5.

I presented several novel techniques for 3D positioning. My work provides benefits for both novice and expert users. Novice users have less experience in 3D modelling, and some have weaker spatial abilities. This motivated me to make my techniques natural and easy for users to understand and learn. Expert users use 3D modelling software more often. They can learn most techniques, yet the novel techniques will still be beneficial to them, as they improve their speed and accuracy in their modelling work. Therefore, I recommend including my techniques in existing and future 3D CAD software. Besides proposing new techniques, I also provide some

guidelines for designing and developing 3D CAD software, for both desktop and VR systems.

1.3. Thesis Structure

This thesis has a cumulative format. In the following chapters, I introduce different aspects of my approaches for better 3D manipulation user interfaces. Chapter 2, 3, and 4 are based on three of my previously published peer-reviewed full papers [Sun et al., 2016] [Sun et al., 2019] [Sun et al., 2018]. They address research questions RQ1 to RQ4. The three published papers are included in these chapters. Chapter 5 includes a study that addresses research question RQ5, and this work is currently accepted in [Sun et al., 2019]. Chapter 6 includes the conclusion remarks.

In Chapter 2, I introduce SHIFT-Sliding and DEPTH-POP, my two new 3D positioning techniques for a desktop system, with mouse and keyboard as input devices. I designed these two techniques based on contact-plane sliding. With SHIFT-Sliding and DEPTH-POP, I generalized contact-plane sliding to floating or colliding objects. The two new techniques significantly outperformed the industry-standard widget-based 3D positioning techniques, even for floating and colliding objects. The work has been published as a full paper titled “SHIFT-Sliding and DEPTH-POP for 3D Positioning” at the ACM Symposium on Spatial User Interaction conference in 2016 [Sun et al., 2016], where it also won a best demonstration award.

In Chapter 3, I extend my work further to invisible objects, i.e., objects that do not meet the visibility assumption of contact-plane sliding. I introduce a new layer-based technique for selecting invisible objects in a desktop environment, that obviates the need to navigate the camera or use wireframe visualization. Users can then select an invisible object from a specific layer of visibility. When sliding an object on an originally hidden surface, I use transparency masks to make the sliding surface visible to the users. The new techniques outperform 3D widgets for positioning of invisible objects. This work has been published as a full paper titled “Selecting and Sliding Hidden Objects in 3D Desktop Environments” at the Graphics Interface conference in 2019 [Sun et al., 2019].

In Chapter 4, I apply the contact-plane sliding techniques in a VR system with an HMD and compared different input devices and cursor displays. The mouse outperforms

the 3DOF controller in a 3D positioning task. The stereo and one-eyed cursors do not have a significant difference in speed or accuracy. The work has been published as a full paper titled “Comparing Input Methods and Cursors for 3D Positioning with Head-Mounted Displays” at the ACM Symposium on Applied Perception conference in 2018 [Sun et al., 2018].

In Chapter 5, I detail my work to further improve 3D manipulation through better positioning techniques, especially for distant objects in VR environments, in systems where the mouse may not be available. I apply the extended sliding technique in a VR system with an HMD using the 3DOF controller as input device. I compare extended sliding and full 3DOF positioning techniques. The results show that extended sliding significantly improves the accuracy of 3D positioning tasks for targets that are in contact with the scene. This work has been accepted as a short paper titled “Extended Sliding in Virtual Reality” at the ACM Symposium on Virtual Reality Software and Technology conference in 2019 [Sun et al., 2019].

Chapter 6 presents my concluding remarks. There, I also revisit the research questions in this thesis and summarize my contributions to the field of 3D manipulation.

Chapter 2.

SHIFT-Sliding and DEPTH-POP for 3D Positioning

In this chapter, I investigate 3D positioning in desktop virtual environments. The motivation for this work is to improve 3D positioning tasks in desktop systems. The results will enable 3D content designers, game designers, or even novice users to benefit from the improved speed and accuracy of 3D positioning tasks. I present two novel techniques that enable precise positioning of 3D rigid objects. As demonstrated by two user studies, both techniques significantly outperform the approach that is the current industry standard. The techniques received very positive feedback from the participants and were perceived to be fast and easy to use.

In 3D virtual environments, users often encounter the need to arrange a scene with numerous objects. Posing a 3D object, i.e., manipulating the position and orientation of an object, is a fundamental task in 3D user interfaces. If the users know the exact target coordinate to position a 3D object, they can easily enter the coordinate of the target position in the system. However, if the object coordinate is unknown to the users, which is usually the case, the users have to select the 3D object with an input device and move the object with some interaction technique, which typically involves moving an input device. This task can be time-consuming, as 6 degrees of freedom (DOF) have to be controlled: 3 DOFs for translation along three axes and 3 DOFs for rotation around three axes. In this chapter, I focus on object positioning (translation).

Some techniques use 3- or 6DOF input devices for object positioning, based on a one-to-one mapping of input and object movement. For such tasks, research has shown that 3DOF input devices outperform 2D devices in some contexts [Hinckley et al., 1997]. Yet, most users are more familiar with the form and function of the mouse. Also, with appropriate mappings, the mouse is generally faster than 3/6DOF devices for accurate placement. Bérard et al. [Bérard et al., 2009] compared the mouse with three 3DOF input devices in a 3D placement task and identified the mouse superior for accurate placement.

In 3D computer-aided design (CAD) systems, the mouse is the most commonly used input device. It has been proven to be a reliable and accurate input device, despite the lack of the ability to directly manipulate a third DOF (or more). To compensate for this shortcoming, various mappings of 2D mouse input to 3D operations have been proposed. For precise 3D object positioning, most CAD user interfaces rely on a local coordinate system to assist object movement controlled with the mouse, typically via 3D widgets [Conner et al., 1992]. In desktop CAD systems, 3D widgets have become the industry standard.

When the user clicks with a mouse on a 3D object to select it in a CAD system, a set of 3D widgets appear on the object. The three axes correspond to the local coordinate system of the object. The users can then click to select any axis and drag the object along the direction determined by that axis. They can also click to select any plane determined by two orthogonal axes and drag the object on the plane. This interaction is a combination of 1DOF and 2DOF movement based on the local coordinate system. Yet, the orientation of this coordinate system is typically independent of other objects in the scene. Moreover, users can only select such widgets after they have selected the object and may have to switch between different parts of the widgets for different 3D movements. This increases the manipulation effort and consumes additional time.

If one moves objects in the real world, the object positions also depend on the surfaces of other objects. Following this concept, some techniques used constraints to help object manipulation, e.g., by snapping the object to geometric features of the scene [Bier, 1990]. Bukowski et al. [Bukowski et al., 1995] used object associations to map the 2D cursor motion into an appropriate object motion in 3D and determine a valid and desirable final location for the objects. Based on the observation of real-world interactions, the contact-plane sliding technique [Oh et al., 2005] linked mouse movement directly to object movement and moves an object on the surface behind it, i.e., uses the contact of the object with that surface. Sliding outperformed 3D widgets in a 3D positioning task.

The contact-plane sliding technique makes three fundamental assumptions: A) The manipulated object stays by default in contact with the rest of the scene, B) The manipulated object does not interpenetrate the scene, and C) The manipulated object is

always at least partially visible to the user. These three assumptions (contact, non-collision, visible) are true for most scenes in the real world.

Contact-plane sliding has its limitations. Objects can only be placed to positions that meet the three underlying assumptions. Since object movement is restricted to maintain contact with surfaces of other objects in the scene, it is not possible to place an object into a floating or colliding position. Also, users cannot easily position an object in depth, i.e., at different positions along the user's view direction.

I introduce SHIFT-Sliding and DEPTH-POP, two new 3D positioning techniques for a desktop system, designed to be used with the mouse and keyboard as input devices. With SHIFT-Sliding and DEPTH-POP, I generalize contact-plane sliding to scenes that do not meet assumptions A or B (contact or non-collision).

With SHIFT-Sliding, I break the contact/non-collision limitation of contact-plane sliding and map positioning to movement based on the coordinate system of the surface that the object was last in contact with, which corresponds to a new form of context-dependent positioning method. The user selects an object under the mouse cursor by a mouse click on the object with the left button. This is a form of image plane selection method [Pierce et al., 1997]. Holding the left button down, they can then drag the object along any surfaces in the scene with mouse movements. During object sliding, the user can press the SHIFT key on the keyboard to move the object orthogonal to the sliding surface, forcing the object to float or collide. When the users release the SHIFT key (with the mouse button still held down), the object will then keep sliding on a plane defined by the initial normal vector. When the users release the mouse button, they "drop" the object at the last position, i.e., it remains where the "drop" occurred.

With DEPTH-POP, I address the inherent depth ambiguity in sliding and enable efficient control of object position in depth. For this, I map discrete mouse wheel actions to object movement in depth, through which the user can then put the object at all those positions along the mouse ray (along the user's view direction), where contact and non-collision assumptions are met. The user first selects an object with a mouse click and slides the object with mouse movement. During object sliding, the users can move the scroll wheel of the mouse forward / backward to move the object further / closer in depth. More specifically, with each wheel action my new DEPTH-POP technique selects the next,

respectively previous, element from the set of 3D positions along the mouse ray that match the three assumptions (contact, non-collision, visible).

During SHIFT-Sliding and DEPTH-POP, the user holds the left mouse button down all the time during manipulation. The manipulated object is always under the mouse cursor and visible, which gives the users better control over the object's position. When the SHIFT key is being pressed, SHIFT-Sliding moves the object continuously along the direction of the normal vector of the current contact surface. In DEPTH-POP, the discrete mouse wheel actions move the object in depth. Together with sliding, both techniques enable users to move objects in all three dimensions directly and independently.

In my system, I map 3D rotation around the objects' center to the right mouse button with wheel operations. If the object is in contact at the start, I maintain all the default assumptions during rotation. If the object rotates to a pose where it would float or collide, I resolve it by moving the object in the direction of the normal vector of the (last known) contact. As my work focuses on object positioning, I did not evaluate the performance of the rotation method in the experiments.

This chapter addresses the first research question: "**How much does extended sliding improve the speed/accuracy of 3D positioning in a desktop system?**" I hypothesize that extended sliding will improve general 3D positioning in a desktop system. Oh et al. [Oh et al., 2005] compared the contact-plane sliding algorithm against 3D widgets in an assembly task for in-contact conditions and found the contact-plane sliding algorithm to be superior. In fact, the 2DOF nature of contact-plane sliding technique matches the affordances of 2DOF input devices such as the mouse. The new SHIFT-Sliding method extends contact-plane sliding to support floating and colliding objects. I hypothesize that SHIFT-Sliding will similarly outperform widget-based techniques for tasks where objects float or collide. I also hypothesize that DEPTH-POP will outperform a widget-based technique. My motivation to choose 3D widgets as baseline in my user studies is compare them with the industry-standard approach.

I had ethics approval for the two user studies I performed, and participants signed consent forms. The results from the two user studies confirmed the hypotheses. Both techniques were significantly faster for 3D positioning compared to the standard widget-based approach. Results from the first study showed that SHIFT-Sliding is easy to

use and, for floating objects, 16% faster than the widget-based approach, the current industry standard. Together with Oh et al.'s results [Oh et al., 2005], this means that SHIFT-Sliding is globally faster than the widget-based method, regardless if the target position is in contact or not. SHIFT-Sliding *automatically* derives a local coordinate system from the last known contact surface, which makes it easy to position objects in space relative to other objects, without having to explicitly set a local coordinate system. The second study showed that SHIFT-Sliding with DEPTH-POP was more efficient, 81% faster, and 67% more accurate than the widget-based technique, as it automatically determines valid object positions in depth. Both techniques profoundly enhanced the ease and speed of 3D positioning with 2D input devices.

I got very positive feedback from the participants, where some even commented along the lines of: "I wish I had this in 3DS Max". I see this as motivation to include the techniques in standard 3D software, as they would help 3D designers improve their speed during 3D modelling. Although I had novice participants in the user studies, I believe that my techniques will benefit both novice and expert users. The details about the two techniques, experimental design, user studies, and results can be found in the following published paper.

I exploit the computing power of GPUs and use the frame buffer for most of the computations. This enables users to slide objects even on complex "surfaces", including point clouds with normal vectors. The technical details of the implementation can be found in the appendix of the published paper.

The work was previously published in a full paper titled "SHIFT-Sliding and DEPTH-POP for 3D Positioning" at the ACM Symposium on Spatial User Interaction conference in 2016. I co-authored this paper with Wolfgang Stuerzlinger and Dmitri Shuralyov.

Abstract

Moving objects is an important task in 3D user interfaces. We describe two new techniques for 3D positioning, designed for a mouse, but usable with other input devices. The techniques enable rapid, yet easy-to-use positioning of objects in 3D scenes. With sliding, the object follows the cursor and moves on the surfaces of the scene. Our

techniques enable precise positioning of constrained objects. Sliding assumes that *by default* objects stay in contact with the scene's front surfaces, are always at least partially visible, and do not interpenetrate other objects. With our new SHIFT-Sliding method the user can override these default assumptions and lift objects into the air or make them collide with other objects. SHIFT-Sliding uses the local coordinate system of the surface that the object was last in contact with, which is a new form of context-dependent manipulation. We also present DEPTH-POP, which maps mouse wheel actions to all object positions along the mouse ray, where the object meets the default assumptions for sliding. For efficiency, both methods use frame buffer techniques. Two user studies show that the new techniques significantly speed up common 3D positioning tasks.

Keywords

3D object manipulation; constraints; frame buffer; layers.

2.1. Introduction

In 3D virtual environments, users often encounter the need to arrange a scene with numerous objects. Here we only deal with the 3D manipulation of rigid objects. Posing a 3D rigid object, *i.e.*, manipulating the position and orientation of an object, is a basic task in 3D user interfaces. This task can be time-consuming as 6 degrees of freedom (6DOFs) have to be controlled: 3 DOFs for translation along three axes and 3 DOFs for rotation around three axes. Some techniques use 3- or 6DOF input devices for object manipulation, based on a one-to-one mapping of input and object movement. For such tasks, research has shown that 3DOF input devices outperform 2D devices in some contexts [Hinckley et al., 1997] [McMahan et al., 2006]. Yet, most users are more familiar with the mouse. Also, in some contexts 2D input is the better choice [Bérard et al., 2009]. As evident by its pervasive use in 3D computer aided design (CAD) applications, the mouse has proven to be a reliable and accurate input device, despite the lack of the ability to directly manipulate a third DOF.

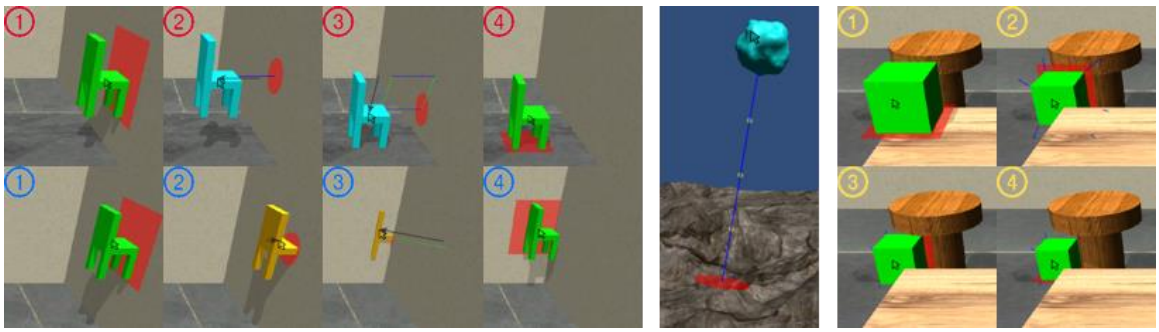


Figure 2.1. The left image rows illustrate SHIFT-Sliding. With this technique the user can lift an object off a surface to float (top row). The object reverts to sliding upon collision. Alternatively, the user can push the object into another (bottom). The object "pops" to the front to avoid being invisible. The middle image shows object height visualization during SHIFT-Sliding. The right image illustrates DEPTH-POP, with a *stationary* cursor. We can place the object into all four positions using up/down mouse wheel actions.

To compensate for this shortcoming, various mappings of 2D mouse input to 3D operations have been proposed. CAD user interfaces use a local coordinate system to assist object movement, typically via 3D widgets [Strauss et al., 1992] controlled with the mouse. However, the orientation of this coordinate system is typically independent of other objects. Still, if one moves objects in the real world, their positions also depend on the surfaces of other objects. Sliding, *e.g.*, [Bukowski et al., 1995] [Oh et al., 2005], links mouse movement directly to object movement and moves an object on the surface behind it, *i.e.*, uses the contact of the object with that surface. This effectively corresponds to manipulation in a *view* coordinate system, independent of the coordinate system of the object. Sliding typically assumes contact and non-collision for manipulated objects. With our enhanced SHIFT-Sliding method we break the contact/non-collision limitation of basic sliding and map manipulation based on the coordinate system of the surface that the object was last in contact with, which forms a new form of context-dependent manipulation.

When multiple surfaces are visible in the same area of a scene, there is an ambiguity in the mapping of 2D input to 3D position, *e.g.*, as the manipulated object could be placed on the table or the floor visible behind it. To address this ambiguity, we introduce a new DEPTH-POP method, which enables efficient control of object position in

depth. For this, we map discrete mouse wheel actions to object movement in depth, which puts the object at all those positions along the mouse ray, where contact and non-collision assumptions are met.

We first review relevant previous object manipulation work. Then we discuss the overall design space and introduce our new interaction methods. In the following, we present implementation details and describe our user studies. Finally, we discuss the results and mention potential future work.

2.2. Related Work

There has been substantial research in the field of object manipulation in 3D user interfaces.

Many mappings of 3- or 6DOF input device movements to object manipulation have been proposed. Ware *et al.* [Ware et al., 1988] introduced the bat, a 6DOF device with a natural one-to-one mapping. Hachet *et al.* [Hachet et al., 2003] introduced the 6DOF Control Action Table, designed for immersive large display environments. The GlobeFish and GlobeMouse techniques [Froehlich et al., 2006] used a 3DOF trackball for 3D manipulation. Bérard *et al.* [Bérard et al., 2009] compared the mouse with three 3DOF input devices in a 3D placement task and identified the mouse superior for accurate placement. Vuibert *et al.* [Vuibert et al., 2015] compared contactless mid-air manipulation with a Phantom and found mid-air manipulation faster but less accurate. Masliah *et al.* [Masliah et al., 2000] studied the allocation of control in 6DOF docking and identified that rotational and translational DOFs are controlled separately. All techniques mentioned in this paragraph require 3D input devices, which currently do not afford the level of accuracy and precision of a modern mouse.

Many touch-based 3D manipulation techniques have been developed. Hancock *et al.*'s [Hancock et al., 2007] multi-touch techniques provide 3D interaction within limited depth. Rotate' N Translate (RNT) [Kruger et al., 2005] offers integrated control of translation and rotation through a single touch-point. Reisman *et al.* [Reisman et al., 2009] presented a screen-space method that provides direct 2D and 3D control. Martinet *et al.* [Martinet et al., 2010] proposed two multi-touch techniques. Users preferred the Z-technique, which permits depth positioning. A later improvement separated translation

and rotation [Martinet et al., 2010]. Herrlich *et al.* [Herrlich et al., 2011] presented two techniques that integrate translation and rotation. Au *et al.* [Au et al., 2012] presented a set of multi-touch gestures for constrained 3D manipulation. In general, the input mappings for touch-based 3D methods require learning and do not support accurate manipulation.

Another approach to 3D manipulation is based on widgets [Conner et al., 1992] [Strauss et al., 1992], which encapsulate 3D geometry and behavior. Such widgets are now prevalent in 3D CAD software. Mine *et al.* [Mine et al., 1997] presented hand-held widgets, *i.e.*, 3D objects with geometry and behavior that appear in the user's virtual hand. Schmidt *et al.* [Schmidt et al., 2008] presented a system that automatically aligned widgets to axes and planes determined by a users' stroke.

Some 3D manipulation systems use 2D input devices, typically the mouse. Bier [Bier, 1990] proposed snap-dragging, which snaps the 3D cursor to object features close to the cursor using a gravity function. Van Emmerik [Van Emmerik et al., 1990] proposed a technique where the user can perform 3D transformations in a local coordinate system through control points. Venolia [Venolia, 1993] presented "tail-dragging", where the user drags an object as it were attached to a rope. With a "snap-to" functionality, other objects also attract the manipulated object. Kitamura *et al.* [Kitamura et al., 2002] proposed a "magnetic metaphor" for object manipulation, which aims to simulate physical behaviors, including non-penetration. In most of these techniques, the local coordinate system for object movement must be explicitly controlled by the user.

Building on Object Associations [Bukowski et al., 1995], Oh *et al.* [Oh et al., 2005] presented a sliding algorithm, where the object follows the cursor position directly and slides on any surface behind it, *i.e.*, the moving object always stays attached to other objects. This form of sliding creates associations automatically, and these associations are not limited to predefined horizontally or vertically aligned surfaces. Compared to click-to-place methods, *e.g.*, [Bukowski et al., 1995], sliding provides better visual feedback as the result of a (potential) placement is continuously visible. For targets in contact, Oh *et al.* compared sliding with axis-widgets and found that sliding is significantly more efficient for novices. Yet, Oh *et al.*'s sliding method lacks direct access to object movement in the third DOF. The authors identified that for some tasks users

have to slide an object on a sequence of surfaces to reach a desired “layer” in depth, which is not always easy to understand, see Figure 2.3.

2.2.1. Contributions

The main contributions we present here are:

- SHIFT-Sliding, which generalizes sliding to support floating and interpenetrating objects.
- A new method to map 2D input to 3D object translation based on the coordinate system of the surface that the object was last in contact with.
- A new DEPTH-POP interaction method that addresses the inherent depth ambiguity in sliding algorithms.
- Comparative evaluations of the new techniques.

2.3. System and Interaction Design

In this section, we discuss the fundamental assumptions that our new object manipulation techniques build on. We target novice users without CAD knowledge. We focus on a desktop-based user interface with a mouse and a keyboard, as this provides high performance in both speed and accuracy. A mouse also helps to keep our system easy to learn and use by novices, as many are used to this interaction device. Yet, the interaction for DEPTH-POP and SHIFT-Sliding is so simple and direct, that it could even be applied to touchscreens, as mentioned in the discussion section.

2.3.1. Design Assumptions

We use a single perspective view, as this corresponds best to how novices are usually presented with 3D content [Wickens et al., 1999]. We do not use stereo, as perspective and occlusion are usually sufficient to accurately judge an object’s 3D position and visibility [Vishton et al., 1995]. In our system, we assume that objects are *by default* in contact with other objects and do not interpenetrate them. Moreover, we

choose not to enable manipulation of objects when they are invisible. Here, we detail the reasoning behind our design assumptions.

1) The manipulated object stays by default in contact with the rest of the scene. As recognized by Teather *et al.* [Teather et al., 2007] and Stuerzlinger *et al.* [Stuerzlinger et al., 2011], floating objects are exceptional on our planet, as (almost) all objects are in contact with other objects in the real world. Also, the exact position of a floating object is harder to perceive accurately, as there are fewer references to judge against. Such objects are also harder to manipulate because more DOFs need to be controlled [Jacob et al., 1994]. In the *default* sliding mode of our system, whenever an object would float, we automatically put the object back into contact with the first surface behind it. With our new SHIFT-Sliding method, we enable the user to override this. When there is nothing behind an object, it will slide parallel to the screen until something appears behind it.

2) The manipulated object does not interpenetrate the scene by default. In the real world, objects do not interpenetrate each other without explicit actions, such as drilling a hole. To avoid unwanted collisions, we choose to avoid interpenetration *by default*. When needed, we permit the user to force an object "into" another with SHIFT-Sliding.

3) The manipulated object is always at least partially visible to the user. Without other forms of feedback (such as haptics) an invisible object cannot be manipulated with precision with normal input devices. Moreover, a common issue in many 3D systems is that an object can become "lost" behind or inside other objects, which then forces the user to "find" it again through navigation. In our system and whenever an object would become completely invisible, we automatically bring the object to a position where it is visible.

When the user selects an object and manipulates it with standard sliding, the object will always remain in positions (and poses) where all three assumptions are met. Beyond the three scene-related assumptions above, we also assume that the scene is rendered as filled polygons. A wireframe representation of objects introduces ambiguities, as it does not explicitly define the enclosed surfaces. This is often difficult to understand for novices. Similarly, we also exclude volumetric content, such as a 3D brain scan.

2.3.2. Basic Sliding

Sliding [Oh et al., 2005] maps object movement so that the manipulated object moves along the surface behind it that it is currently in contact with. We use the normal vector at the contact point to determine the sliding plane. With this contact constraint, we can directly map 2D motions of the mouse cursor to 3D movement of the object. Figure 2.2 illustrates sliding. When the user selects an object (at position A), we record the intersection of the mouse ray on the object as the start point. We identify the normal vector at the contact point on surface 1 via the frame buffer. The start point and the normal vector define the sliding plane. The intersection of a new mouse ray and the sliding plane becomes the end point of the object translation. By moving the mouse cursor, the user then effectively translates the object parallel to the sliding plane. We examine the situation with multiple contacts in different planes in the discussion section.

When the user slides the object to position B in Figure 2.2, any further upwards movement would cause the object to float. Here, basic sliding snaps the object back to the next surface behind it, to position C on surface 2, while keeping the object under the mouse cursor. The new contact point provides a new normal vector for a new sliding plane. Then the user can slide the object on surface 2, to positions such as D. If the user now moves the mouse back down, the object can reach position E, where it is still partially visible from the camera. Yet, if the user slides the object from E further downwards, the object would become invisible to the user. Here, we *"pop" the object to the front*, *i.e.*, bring it to a position below (and a bit to the left of) B, in contact with surface 1. With this method we keep the mouse cursor at the same point on the object throughout, as an additional cue for object position, giving the user also a better understanding of the 3D object movement. To highlight the fact that the object is in contact, we render a semi-transparent rectangle at the contact surface.

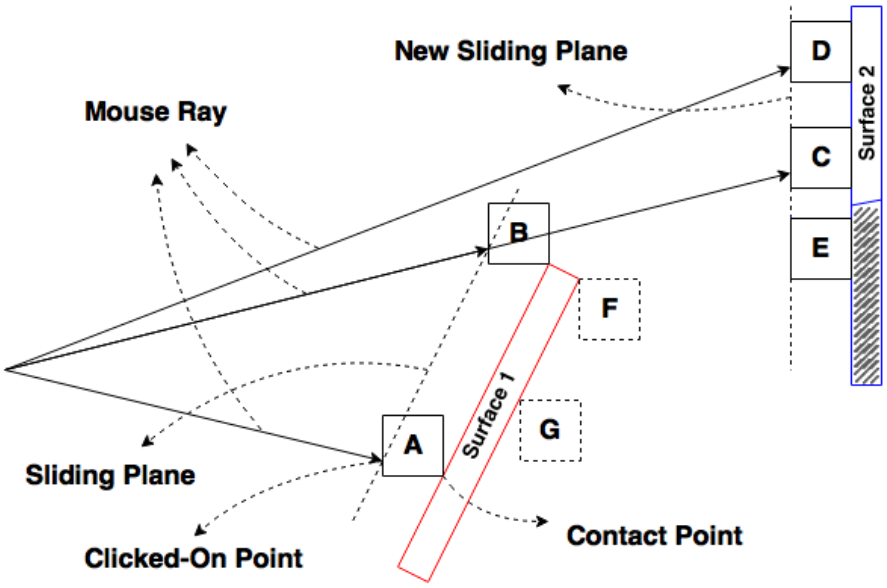


Figure 2.2. Illustration of sliding movements for an object across the front surfaces of two objects with an upwards mouse movement (positions A-D). The shaded part of surface 2 is occluded by surface 1. Position E can only be reached from C with a downwards mouse movement. Positions F and G cannot be reached from the current viewpoint, see text.

When dragging an object along a surface nearly aligned with the view direction close to the horizon, it can easily disappear into the distance, since a small mouse movement can cause a dramatic object movement in 3D. To deal with this, we perform an occlusion test for every frame and bring the object to the front whenever it becomes completely invisible — even if this causes the object to slide on another surface. When there is no surface behind the object, we keep the object sliding parallel to the screen and inside the camera frustum (even if the projection of the object is very small). With this, the user can never lose sight of the moving object.

In our system, the user can slide objects on invisible surfaces, such as a portion of the surface 2 in Figure 2.2 that is hidden by surface 1 (near E). To keep the interaction intuitive, we only permit sliding on surfaces facing the user. If we were to enable sliding on back-faces, this would create inappropriate mappings. Consider that the user slides from A to B with an *upwards* movement of the cursor along surface 1 in Figure 2.2. To reach position F, the cursor movement would then need to be mapped to

a *downwards* motion of the mouse, which is indistinguishable from a movement back towards A. Thus, a better option is to snap the object to a position beyond surface 1 and slide the object on surface 2.

Object sliding maps mouse input to 3D object motions, which automatically defines an appropriate local movement plane. This also guarantees that the cursor and object always stay aligned, regardless of the surface involved. The new SHIFT-Sliding technique *generalizes* sliding to situations where objects float or interpenetrate. Our new DEPTH-POP technique gives the user discrete control over object depth relative to the camera with a contact assumption.

2.4. SHIFT-Sliding

Basic sliding keeps the object in contact with the scene and automatically chooses the local sliding plane for the object. For some use cases, such as 3D game design or animation, there are scenarios where some objects are floating. Imagine a car in an action sequence or a bouncing ball in a 3D game. Basic sliding cannot deal with those situations effectively. With SHIFT-Sliding, users can break the assumptions of contact or non-collision. We still pop the object to front if it would become invisible to ensure accurate manipulation and for consistency.

While manipulating an object, the user can move the object *orthogonal* to the sliding plane by pressing the SHIFT key. If the user then “pulls” the object away from the surface, this will cause the object to float, see Figure 2.1. When the user releases the SHIFT key (with the mouse button still held down), the object will then keep sliding on a plane defined by the initial normal vector. In this state, the floating check is temporarily disabled. To provide feedback, we highlight the moving object in a different color in SHIFT mode. When the floating object collides, we transition the object back into sliding mode and start sliding on the collider surface.

If the user lifts an object up and releases the mouse button, the object floats and is highlighted accordingly. While the object is still highlighted, the user can later “re-capture” the object with a click, is then back in SHIFT-sliding mode, and can move the object on the previous sliding plane. If a floating object is no longer highlighted, it will snap back to a surface when clicked and then start sliding. Alternatively, with another

SHIFT-click, the user can move the object up or down orthogonal to the sliding plane. This is an improvement over Object Associations [Bukowski et al., 1995], where breaking an association leads only to a three-axis manipulation mode.

If the user "pushes" the object into a surface while pressing the SHIFT key, the object will interpenetrate that surface. When the user releases the SHIFT key, the object will then keep sliding on the plane defined by the original normal vector, inside the surface, still maintaining the visibility assumption. We temporarily disable the collision check for objects pushed into a surface.

2.5. DEPTH-POP

Moving the object in the 3rd dimension with a 2D input device involves indirect mappings. With sliding, previous work has observed that users can move objects along the "shortest continuous path across visible surfaces," through long mouse motions [Oh et al., 2005]. For example, in Figure 2.3 and to move an object from the wall to the table top, novices will typically slide the object along the wall, the table leg and then onto the table surface [Oh et al., 2005]. With DEPTH-POP, the user can accomplish the same result with a *single* mouse wheel action, which makes manipulation more direct. To achieve the same result, Object Associations [Bukowski et al., 1995] or other "click-to-place" algorithms require the user to first move the object away and then into the right place, which requires a minimum of two move operations.

Hinckley *et al.* [Hinckley et al., 1994] used discrete cycling to select multiple objects along a mouse ray, but did not use this for positioning an object in 3D. LayerPaint [Fu et al., 2010] permits drawing continuous strokes even on occluded regions in multi-layer scenes through automatic depth determination. Igarashi *et al.* [Igarashi et al., 2010] presented layer swap, which allows the user to directly modify the depth order of the top two layers by clicking on a 3D layered object. They also presented layer-aware dragging. While dragging an object, the user can toggle between drag-over and drag-under modes with the SHIFT key and the system will adjust the object's depth automatically.

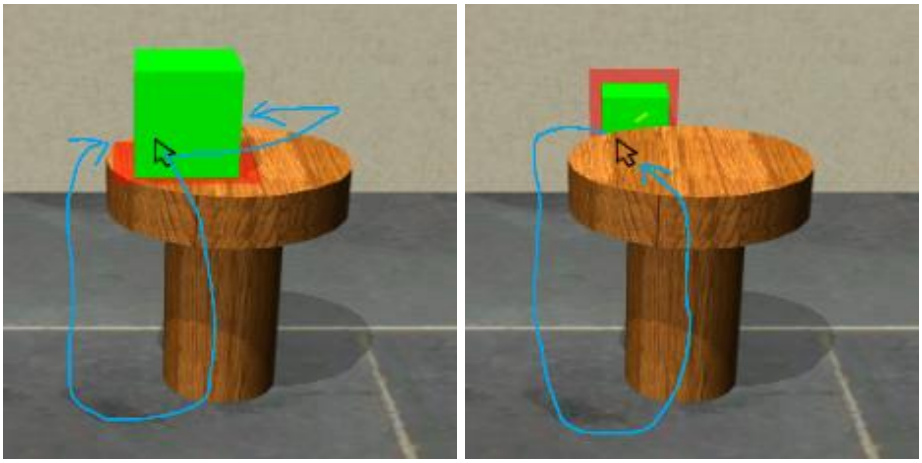


Figure 2.3. With sliding one can move between the two object positions by following one of the blue mouse paths. For the right path in the left image, the object snaps to the wall when leaving the table. With DEPTH-POP the user can directly transition with a *single* mouse-wheel action, without moving the cursor.

Extending these works, we map the choice of 3D object position to scroll wheel actions, whenever the user is dragging/sliding an object with the left button. More specifically, with each wheel action our new DEPTH-POP technique selects the next, respectively previous, element from the set of 3D positions along the mouse ray that match our assumptions (contact, non-collision, visible). We map front and back movement of the mouse wheel to “push-to-back” respectively “pop-to-front”. Together with sliding, this enables users to move objects in all three dimensions directly and independently.

2.5.1. Push-to-Back/Pop-to-Front

Moving the mouse wheel away from/towards the user triggers a push-to-back/pop-to-front event, *e.g.*, between position B and C in Figure 2.2. With push-to-back the object is moved to the next possible position further away from the camera that satisfies all three main design assumptions. For each pop-to-front event, the object is moved to the next position closer to the camera again maintaining the design assumptions. We also call pop-to-front whenever the object becomes completely occluded, *i.e.*, invisible.

2.5.2. Audio Feedback

For DEPTH-POP actions, we use different auditory cues to indicate a successful DEPTH-POP or a failed attempt to give the user feedback. Examples of infeasible actions are an object that is already the foremost visible object and thus cannot be pulled closer or an object to be pushed further away but with nothing behind it.

2.5.3. Orthographic vs. Perspective Projection

We display the 3D scene in perspective and use that camera also for visibility detection. After all, when all pixels of the selected object are invisible, the user cannot see and manipulate it with precision. If our system detects this situation, we call pop-to-front.

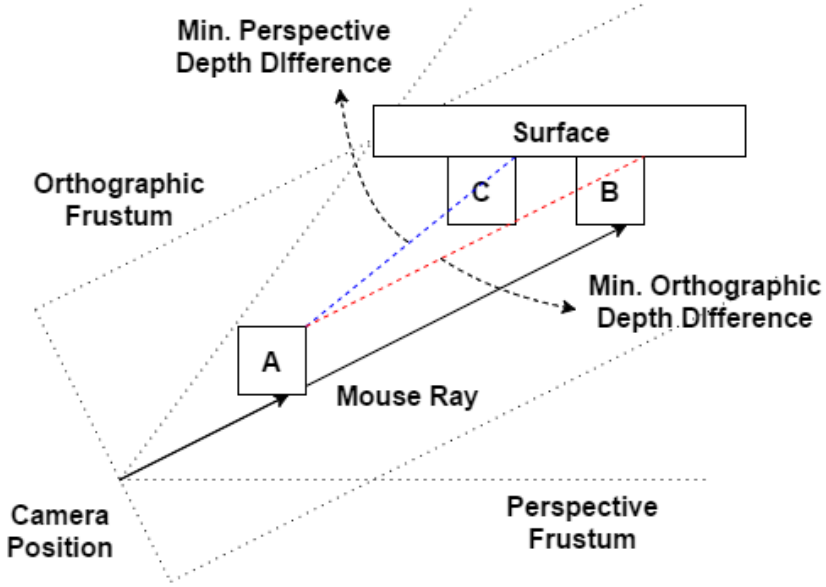


Figure 2.4. Perspective vs. orthographic projection. In orthographic projection, an object at position A will be popped to B where the minimum depth difference occurs (red dashed line), keeping the cursor stable. In perspective, the smallest depth difference (blue dashed line) would move the object to C, violating the static cursor property.

When we move the object along the mouse ray to bring an object closer or further away we use an orthographic camera in the DEPTH-POP algorithm. This design

decision makes a functional difference for the user, as it guarantees that the point on the selected object remains stable underneath the mouse cursor at all times, which yields a better interaction mapping. After all, if the cursor position on the object shifts/changes during sliding, object movement becomes less predictable, and thus less precise. See Figure 2.4. The DEPTH-POP algorithm (see the appendix) computes a distance in depth, which corresponds to the distance that the object should move along the mouse ray. If we were to use a perspective camera, the minimum difference in depth occurs along various rays (a different one for each pixel), rather than a specific direction. Thus the smallest perspective depth difference is different from the mouse ray direction, which in turn would violate the static cursor property. Orthographic projection does not suffer from this ambiguity. Also, linear movements in orthographic projection correspond better to how an object moves in 3D. Thus, we set up an orthographic camera in the direction of the mouse ray and use the frame buffer of this camera for all computations and DEPTH-POP.

2.6. Implementation

Here we discuss implementation details of our system. We built our system in the Unity game engine. We use a desktop computer with 3.5 GHz i7 processor, 16 GB of memory, and two NVIDIA GeForce GTX 560 SLI graphics cards. We use a mouse and a keyboard as input devices.

2.6.1. Frame Buffer vs. Geometry

We exploit the computing power of GPUs and use the frame buffer for most of the computations. This also enables us to slide objects on more complex “surfaces”, including even point clouds with normal vectors. Geometry-based methods would suffer from decreased performance with complex objects and surfaces. To support non-convex geometries, we use depth peeling [Bavoil et al., 2008] to compute all depth values for the hidden layers of the scene. With depth peeling, each unique depth layer in the scene is extracted, and the layers are enumerated in depth-sorted order. As in other systems, a left mouse down selects the closest object along the mouse ray. We highlight the selected object with a different color during sliding.

2.6.2. Floating and Collision Checks

When the user selects an object that is not in contact with a surface, or when an object slides off a surface, we need to push the object back to force it into contact with the scene. When the object collides with the scene, we pull the object front to resolve the collision.

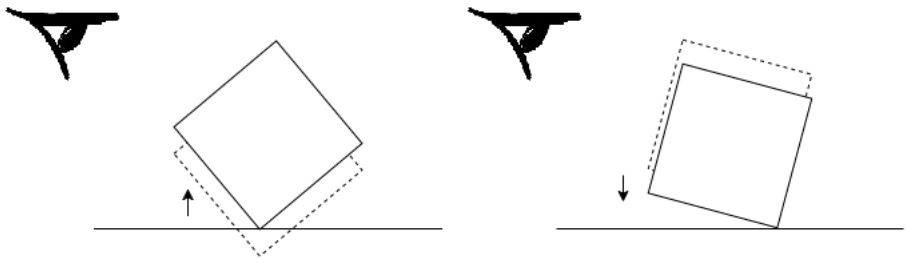


Figure 2.5. During rotation, an object is always kept in contact with the surface. To ensure contact, we move the object in the direction of the (contact) normal vector to resolve floating and collision situations.

After floating and/or collision is resolved, we slide the object as previously described. Our floating and collision checks guarantee that the object is always in contact with the scene, and that the sliding plane changes seamlessly.

2.6.3. Contact-Based Object Rotation

We map 3D rotation around the objects' center to the right mouse button with the two-axis valuator method [Bade et al., 2005] and wheel operation in this state to rotation in the 3rd dimension [Shuralyov et al., 2011]. If the object is in contact at the start, we maintain all our design assumptions during rotation. If the object rotates to a pose where it would float or collide, we resolve it by moving the object in the direction of the normal vector of the (last known) contact. See Figure 2.5.

2.7. Evaluation

We first evaluated the technical performance of our system. The system runs stably at 60 fps for scenes with almost a million polygons. The user can slide objects on

various surfaces, including concave surfaces and point clouds. For the scenes shown on the right in Figure 2.6, a single DEPTH-POP operation takes less than 20 ms. We conducted two user studies to evaluate our new techniques.

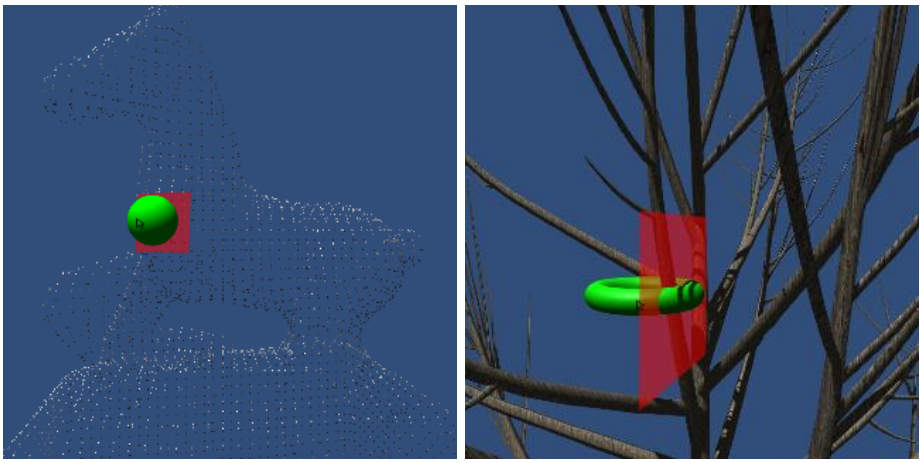


Figure 2.6. In the left image, the object slides on a point cloud. On the right, the torus can be placed on or around any branch of the tree (assuming enough space to avoid collisions).

2.7.1. Evaluation of SHIFT-Sliding

Oh *et al.* [Oh et al., 2005] had compared the Sliding algorithm against 3D widgets in an assembly task for in-contact conditions and found the sliding algorithm to be superior. Our new SHIFT-Sliding method extends basic sliding to support floating and colliding objects. We hypothesize that our new method will similarly outperform widget-based techniques for tasks where objects float or collide. To address a potential confound we *disabled camera navigation* in our study. To ensure internal validity, we also disabled object rotation and DEPTH-POP for this study.

In a pilot study we compared our sliding technique against widgets in various conditions, including single and four-view presentation. In the 3D widgets method, the user can drag either the three axes manipulators or the corresponding plane manipulators to move the object in one or two dimensions. For all situations where objects were in contact, we observed that our results matched the outcome of the previous evaluation of sliding [Oh et al., 2005]. Average sliding and widget times were significantly different ($F_{1,11} = 91.92, p < 0.001$) with 6.05 seconds, respectively 30.70,

which matches the main result of Oh *et al.* [Oh et al., 2005]. Thus, we examine in our first user study only the manipulation of objects not in contact with the scene, *i.e.*, floating objects.

2.7.2. User Study 1

For floating objects, widget-based manipulation is easier if the local coordinate system of the object aligns with the world system. This can affect performance substantially. Thus we investigate coordinate system alignment through task subsets in our study.

Apparatus & Participants

We used the implementation described above to conduct this experiment. We recruited 12 (5 female) undergraduate and graduate students from the local university population. We did not screen participants for 3D/gaming experience. Our participants had varying game expertise, with 58% being regular gamers and 42% playing games only rarely. There was a 5-minute training session before the study, which introduced participants to the techniques in a playground environment, but did not include any version of the experimental tasks.

Experiment Design

We designed a 3D object positioning experiment and asked participants to move an object to a target position in various scenes. When the user positioned the object in the target, we measured the completion time and relative distance from the ideal target position. We recorded all actions of each user. The experiment had a 2 (techniques) x 2 (displays) x 2 (alignment) design. The order of technique, display, and alignment conditions was counter-balanced to avoid learning effects. The first technique uses a 3-axis widget aligned to the local coordinate system of the object. We call this technique LCS. With LCS, the user can drag either the three axes or the corresponding planes to move the object, as in most 3D editing software. The second technique is our new SHIFT-Sliding algorithm. We call this technique SHS here for brevity. The first display condition used four views (one perspective and three orthogonal views), corresponding

to the standard user interface in 3D editing software. The second condition uses only a single perspective view. Figure 2.7 top shows the SHS condition with four views.

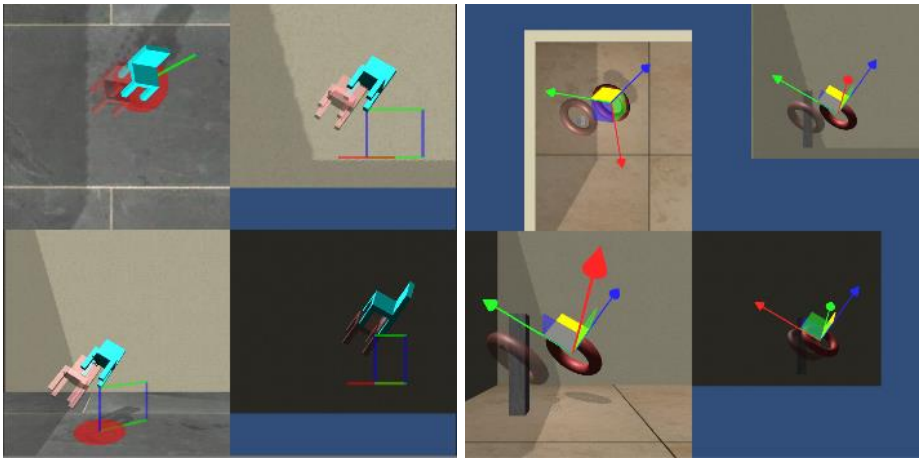


Figure 2.7. Top: The SHS condition with four views. The one-view condition uses only the bottom left view in full screen. The transparently shown target pose is floating above the floor. Bottom: The LCS condition with four views. The target position is around the pillar.

As discussed before, we designed our experiment to focus on floating objects. To investigate the effect of object alignment with the scene, the tasks were composed of two subsets, corresponding to aligned or rotated poses relative to the world coordinate system. The object orientations in the aligned condition were aligned with the three axes in the world coordinate systems. In the rotated condition, objects were rotated 45 degrees on all three axes relative to the world coordinate system. The effect of such object alignment had not been investigated in previous work. Each task condition had 5 trials, with different objects and scenes. The target positions were positioned so that movement along all three axes was necessary. On average the movement distance along each axis corresponded to a third of the viewport size (in the orthogonal views). Each user performed all trials in both two task conditions with all techniques and displays, corresponding to a total of 40 (5x2x4) trials for each user. We asked the participants to perform the tasks as quickly and as accurately as possible.

User Study Results

We used linear mixed models (with repeated measures) to incorporate subject variability. A critical value $\alpha = 0.05$ was used to assess significance. The results showed

that SHS (M=33.31 seconds, SE=1.85) is significantly faster than the industry standard LCS (M=38.56, SE=1.88), OneView (M=31.37, SE=1.55) is significantly faster than FourView (M=40.50, SE=2.12), and the aligned condition (M=31.99, SE=1.71) is significantly faster than the rotated one (M=39.89, SE=2.00). See Table 2.1 and Figure 2.8.

Table 2.1. Linear mixed model analysis results for completion time and distance for study 1.

Source	$\chi^2(1)$ time	Sig	$\chi^2(1)$ distance	Sig
Tech	4.73	*	0.30	ns
View	14.32	***	19.24	*
Align	10.72	**	0.56	ns
Tech*View	4.41	*	1.30	ns
Tech*Align	17.33	***	0.01	ns
View*Align	6.44	*	1.25	ns
Tech*View*Align	15.60	***	0.11	ns

ns/ms = not/marginally sig., *,**,*** = p<0.05,0.01,0.001.

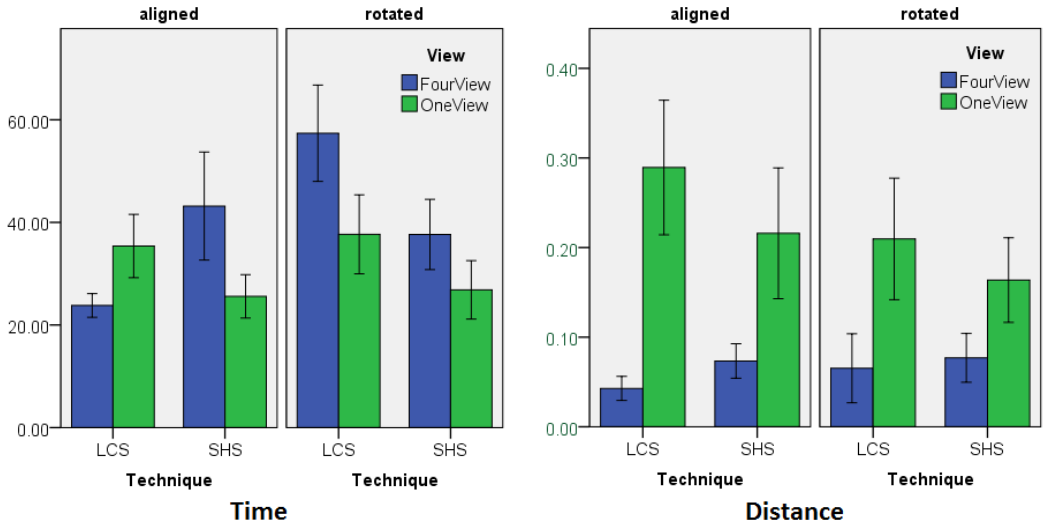


Figure 2.8. Time and distance for study 1. Error bars show 95% confidence intervals.

In terms of completion time, all interactions were significant: SHS-OneView is faster than SHS-FourView and LCS-OneView. SHS-rotated and LCS-aligned are both

faster than LCS-rotated. FourView-aligned and OneView-rotated are both faster than FourView-rotated.

A Tukey-Kramer's test shows that for *aligned* targets, SHS-OneView (M=25.58 seconds) is not significantly different from LCS-FourView (M=23.80). For rotated targets, SHS-OneView is significantly faster than all other combinations.

In terms of target distance, technique did not have a significant effect. FourView (M=0.065, SE=0.007) had a significantly smaller distance than OneView (M=0.220, SE=0.017). Alignment did not have a significant effect. There were no significant interactions on target distance.

Nine of 12 participants found SHS easy to use. Eight participants prefer the SHS technique over LCS. Additionally, we got very positive feedback, see the discussion. Those who did not prefer SHS stated that the need to hold SHIFT down makes coordination slightly harder, but more practice might help.

2.7.3. Evaluation of DEPTH-POP

We performed another pilot to compare basic sliding with DEPTH-POP. Several participants aborted the study due to frustration with basic sliding which made results hard to interpret. In scenes with simple geometry, such as Figure 2.3, sliding "around" works reasonably well, once users figure this out. Yet, in scenes with small thin features such as Figure 2.6 right, moving the torus between depth layers is challenging, as one cannot rely on the object becoming invisible to pop the object to the front. One viable strategy is to purposefully force a collision to pop the object to the front; yet then it becomes impossible to place the torus around the branch again. An alternative way for placing it around the branch is to slide an object from the tip of a branch inwards, but none of our participants were able to figure this out. Thus we concluded that basic sliding is not suited for such scenes and did not evaluate it.

2.7.4. User Study 2

In our second study we evaluated the DEPTH-POP algorithm in isolation, for objects in contact with the scene. We hypothesize that with DEPTH-POP users would be able to complete tasks quicker than with a widget-based technique.

Apparatus & Participants

We evaluated our implementation of DEPTH-POP in this experiment. We recruited 12 (5 female) different graduate and undergraduate students from the local university population. We did not screen participants for 3D/gaming experience. Our participants had varying game expertise, with 42% being regular gamers and 58% playing games only rarely. There was a five-minute training session before the study, which introduced participants to the techniques.

Experiment Design

Similar to the first study, the experiment had a 2 (techniques) x 2 (displays) x 2 (alignment) design. The order of technique, display, and alignment conditions was counter-balanced to avoid learning effects. We used LCS again as the first technique. The second technique was SHIFT-Sliding with DEPTH-POP enabled, which we call SDP here. The display conditions were again four views and a single view. We also looked again at tasks with aligned and rotated poses. All tasks involved only objects in contact and could be achieved with basic sliding and DEPTH-POP, *i.e.*, did not require SHIFT-Sliding. Still, we did not disable SHIFT-Sliding, as some tasks can indeed be completed with SHIFT-Sliding and the automatic collision response of our system. We had again 5 trials for each object alignment condition. Each user had to perform 40 (5x2x4) trials. Figure 2.7 bottom shows the LCS condition with four views.

User Study Results

The results of a linear mixed model (with repeated measures) analysis showed that SDP (M=17.55 seconds, SE=1.18) is significantly faster than LCS (M=31.85, SE=1.57), OneView (M=22.41, SE=1.24) is significantly faster than FourView (M=27.00, SE=1.64), and the aligned condition (M=18.33, SE=0.97) is significantly faster than the rotated condition (M=31.07, SE=1.73). In terms of completion time, the interaction

between views and rotation was significant: FourView-aligned and OneView-rotated are both faster than FourView-rotated. See Table 2.2 and Figure 2.9.

Table 2.2. Linear mixed model analysis results for completion time and distance for study 2.

Source	$\chi_2(1)$ time	Sig	$\chi_2(1)$ distance	Sig
Tech	71.67	***	8.80	**
View	7.37	**	36.86	***
Align	56.83	***	0.58	ns
Tech*View	1.11	ns	9.55	**
Tech*Align	3.56	ms	0.03	ns
View*Align	13.16	***	1.77	ns
Tech*View*Align	7.68	**	0.20	ns

ns/ms = not/marginally sig., *,**,*** = p<0.05,0.01,0.001.

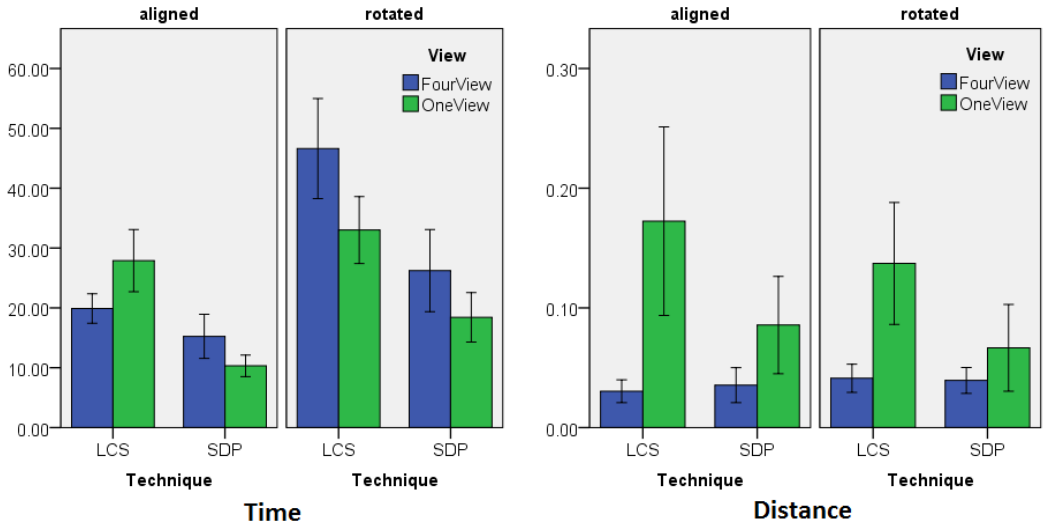


Figure 2.9. Time and distance for study 2. Error bars show 95% confidence intervals.

In terms of target distance, SDP (M=0.057, SE=0.007) had a significantly smaller distance than LCS (M=0.095, SE=0.012). Moreover, FourView (M=0.037, SE=0.003) had significantly smaller distance than OneView (M=0.115, SE=0.014). Alignment did not have a significant effect. The interaction of technique and view had a significant

effect. Both LCS-FourView and SDP-OneView had a significantly smaller distance than LCS-OneView. The other interactions were not significant.

Seven participants found the SDP technique easy to use and relied solely on DEPTH-POP to complete the tasks. A few participants found it harder to understand when to use DEPTH-POP, so they performed *some* of the trials with SHIFT-Sliding. They stated that with more practice they would use it more frequently.

2.7.5. Improvements to 3D Object Position Visualization

Based on the feedback from the participants as well as our observations, we added some visualizations to our system to enhance the perception of 3D positions. Some users found it hard to judge the sliding plane and movement relative to the lift position in the floating state with SHIFT-Sliding. In DEPTH-POP, some users had issues with the idea that object movement is along the cursor ray. To address these issues, when an object is lifted up, we now draw a line with markers equal to the bounding box size (projected in the normal direction) to indicate height in SHIFT-Sliding, see Figure 2.1. When the user slides an object away from the initial lift position, we show additional lines in the local coordinate system that connect the object's current and lift position, see Figure 2.7 top. This provides strong perspective cues, which further help the user to better judge the object's position in 3D. To clearly indicate that an object floats, we replace the semi-transparent rectangle for contact visualization with a small circle. Unlike interactive shadows [Herndon et al., 1992], this circle is not interactive. For push-to-back DEPTH-POP actions we also show (dark blue) guides to help the user understand the 3D movement better, see Figure 2.1.

2.8. Discussion

Sliding keeps the manipulated object by default in contact with the remainder of the scene. The assumption is true for most scenes in the real world, and thus facilitates object movement in many scenes. SHIFT-Sliding adds the new ability to have objects float or interpenetrate. Moreover, SHIFT-Sliding *automatically* derives a local coordinate system from the last known contact surface, which makes it easy to position objects in space relative to other objects, without having to explicitly set a local coordinate system.

Results from the first study show that SHIFT-Sliding is easy to use and, for floating objects, 16% faster than the widget-based approach, the current industry standard. Together with Oh *et al.*'s results [Oh et al., 2005], this means that *SHIFT-Sliding* is globally faster than the widget-based method, regardless if the target position is in contact or not. Given the frequency of widget-based positioning in the 3D workflow, this means that *SHIFT-Sliding* can result in substantial time savings for practitioners. We got very positive feedback from the participants, where some even commented along the lines of: "*I wish I had this in 3DS Max*". Moreover, SHIFT-Sliding in a single perspective view is never significantly worse than the widget-based approach. With SHIFT-Sliding, users received enough depth cues to complete the tasks in the single perspective view. They found it easy to find an appropriate plane to start sliding with the SHIFT key, even for complex surfaces. In fact, we observed that it does not matter that much where in a given 3D movement task users start to use the SHIFT key to lift the object. For rotated target poses, SHIFT-Sliding was at least 29% faster than any widget-based condition. Widget-based manipulation also suffered with rotated targets, as dragging in the (rotated) widget coordinate system causes movement in more than one direction in the world coordinate system, which is harder to understand for users. Thus, SHIFT-Sliding effectively *merges the freedom of widget-based manipulation with the efficiency of sliding*. This fundamentally improves 3D object manipulation with 2D input devices.

The second study shows that SHIFT-Sliding with DEPTH-POP is more efficient, 81% faster, and 67% more accurate than the widget-based technique, as it *automatically* determines valid object positions in depth. The (seemingly) simple mapping to mouse-wheel actions together with our accelerated implementation greatly simplifies moving objects in depth and radically accelerates the associated tasks. DEPTH-POP facilitates even more challenging tasks, such as fitting a complex object around another one.

Results from both studies show that the OneView condition is on average more efficient than FourView, but less accurate. Based on our observations during the study, users might have tried to be more accurate with FourView display, at the expense of efficiency. Also, with widget-based manipulation, FourView was faster for aligned targets, yet OneView faster for rotated targets. We are not certain that this last finding holds up, as in the orthogonal views of the FourView widget condition a *third* of the axis controllers did not always work correctly with rotated objects in the study. While users were able to complete the tasks, we recognize that the faulty controllers may have had a

limited negative effect for this specific condition. Yet, participants had experienced this issue in the training session and thus learned to use the other views and/or controllers. Moreover, based on our observations during the study, we believe that the impact of this issue was overshadowed by the fact that users struggled (much) more with the challenges posed by a locally rotated 3D coordinate system.

In both studies, the participants commented that more practice would help. It would be interesting to measure the learnability of our new methods in a long term study. To address potential issues around having to hold the SHIFT key down during SHIFT-Sliding, one option would be to use the SHIFT key similar to a toggle [Igarashi et al., 2010].

2.8.1. Other Reflections

As there are only two interaction modes in our system, we can easily adapt our technique for a *touchscreen* interface. As is standard in most touch interfaces, the user can select and slide an object with a single finger. A second touch/hold can then serve as an activation event for SHIFT-Sliding to float an object. A *flick* of a second finger can be mapped to DEPTH-POP actions, depending on the direction of said flick. The second finger could be either a finger of the other hand or the same hand, as in recent work [Scheurich et al., 2013].

If the selected object has multiple contact points, the sliding behaviour depends on the contact points and their normal vectors. If the object slides across two identical table surfaces that are positioned side-by-side, the normal vectors and contact planes are the same. Thus the object can smoothly slide from one table to the other. If the object slides on a table surface towards a wall, the normal vector for the new contact will be different. In this case, the object would collide and we pop the object to the front (of the wall). Then, it will slide on the wall. Previous work, *e.g.*, [Oh et al., 2005], has already shown that multiple (compatible) contacts can be used to slide an object.

In our system users control translational and rotational DOFs separately during manipulation [Masliah et al., 2000]. We choose to keep object orientation static during sliding. Alternatively, we could also dynamically change the objects' orientation with the

normal vector of the sliding plane, thus keeping the same surface of the manipulated object in contact with the scene. This could be provided as a separated mode.

Ayatsuka *et al.* [Ayatsuka et al., 1996] already identified that manipulation via interactive shadows is unnatural, since three projections are needed. Yet, their penumbrae method [Ayatsuka et al., 1996] has also the drawback that penumbrae scale non-linearly with object height. The shadows in our system are not interactive. In the work of Glueck *et al.* [Glueck et al., 2009], the shaded inner region of the base coarsely indicates the height of objects. Yet, as reported by Heer *et al.* [Heer et al., 2010], circular area judgments are not that accurate. The length of their "stalks" shows object heights directly – but is still perspectively foreshortened. We instead show markers at regular intervals to facilitate quick perception of object height.

As mentioned before, we disable back-face sliding. Yet, we could also temporarily enable such sliding in our system in some situations. One potential scenario is to permit the user to put an object in contact with a back surface with DEPTH-POP and slide along it. This would not cause the inappropriate mapping issue discussed before. Whenever the object loses contact, *i.e.*, starts floating or collides we would then transition to basic "front" sliding.

3D scanning a real scene yields point clouds. Converting such point clouds to geometry requires extra work. If the application scenario requires the user to place synthetic objects into a scanned scene, sliding can be used directly on the point clouds. For this, we only need normal vectors for each sample point. Then the contact point and sliding plane can potentially change at every frame during slides. SHIFT-Sliding also works on point clouds. However, DEPTH-POP only works if the samples form reasonably dense layers. With DEPTH-POP it is possible to place the object inside the "body" of a point cloud, *i.e.*, locations where the normal vector of the contact point is pointing away from the user. To avoid this, it is better to limit sliding only to front-facing points.

We accelerate most of the computations through the GPU. We use the frame buffer for most of the components in the system, including collision detection. For simplicity, we chose to use an image-based technique for identifying collisions, but any method, which provides information about the position, normal vector, and interpenetration distance where the collision occurs, suffices.

2.9. Conclusion

We presented two novel 3D positioning techniques that are efficient and easy to use. We extend basic sliding with the new SHIFT-Sliding and DEPTH-POP methods. The results of the user studies showed that for novice users the new methods are more efficient for 3D positioning compared to the standard widget-based approach. Both methods profoundly enhance the ease and efficiency of 3D manipulation with 2D input devices.

In the future, we plan to explore if rendering front layers transparent can aid the manipulation of invisible objects, based on previous work [Fu et al., 2010] [Ortega et al., 2014]. Also, we intend to look at new methods for manipulation with multiple constraints.

The current implementation of DEPTH-POP can slow down in scenes with high depth complexity on lower-end graphics hardware. In scenes with many hidden layers, the large amount of small fragments could lead to a huge amount of solutions. Many of those solutions might be meaningless for interaction. In the future, we will optimize the algorithm to deal better with such cases through appropriate pruning.

2.10. Acknowledgements

We would like to thank all the participants.

2.11. Appendix

In this appendix, we describe the technical details of our new interaction methods, starting with several helper algorithms and implementation details.

2.11.1. Depth Intervals

To enable DEPTH-POP, we first introduce a new algorithm based on depth intervals. The algorithm for depth intervals will output the optimal distance for the object to be pushed to the back or popped to the front. For simplicity of explanation, we focus

our discussion here on convex objects and generalize the algorithm to concave cases later.

We use an *orthographic view* as this corresponds better to the way the object moves. First, we render the moving object twice to generate a depth buffer image for both the front and back-facing surface of the object, which defines the “thickness” of the selected object at each pixel. We limit the size of the 3D orthographic view frustum to the bounding volume of the object to maximize precision and minimize computation.

```
void MergeInterval() {
  MergedIntervalSet = {};

  for every overlapping pixel {
    LowerBound = SceneDepth - ObjectBackDepth;
    UpperBound = SceneDepth - ObjectFrontDepth;
    CurrentInterval = [LowerBound, UpperBound];
    if(CurrentInterval overlaps an interval T
                  in MergedIntervalSet) {
      Merge(CurrentInterval, T);   }
    else MergedIntervalSet.Add(CurrentInterval);
  }
}
```

Figure 2.10. Pseudo code for merging intervals for push-to-back.

Next, we create an empty interval set and start to populate it. We loop over all pixels where the selected object and the scene overlap. For each pixel, we calculate the difference of depth values between the front and back surface of the object and the front surface of the scene. The depth difference between the scene and front-facing and back-facing surfaces of the object yield the upper and lower bound of each depth interval. Depending on the desired movement direction, one of the two interval bounds indicates the correct amount of movement, to get the selected object back into contact with another surface. For each pixel each depth interval is added to the interval set, merging with existing intervals as appropriate. Each interval in the fully merged interval set for a pixel then corresponds to a possible *global* solution for pop-to-front or push-to-back. We then combine the solutions across all pixels with a minimum (or maximum)

search to identify the globally best solution. Figure 2.11 illustrates the merged depth interval set for push-to-back.

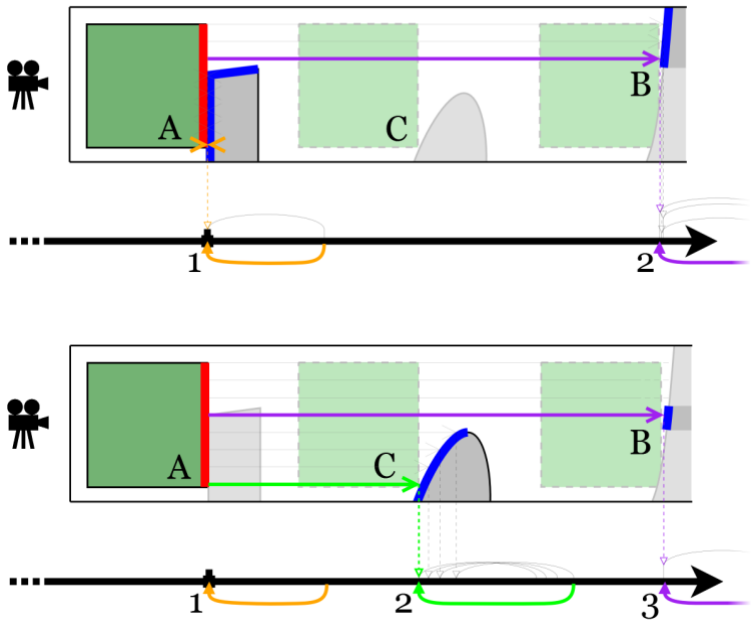


Figure 2.11. Merged depth interval set for the scenario of push-to-back on the *first* scene layer (top image) and second *layer* (bottom). In both images the lower bound of interval 1 corresponds to the current object position. At the bottom the lower bound of interval 2 moves the object to position C.

2.11.2. DEPTH-POP for Convex Objects

In the push-to-back algorithm, for the depth interval at each pixel, the lower bound is the depth difference between the object back-facing layer and a front layer scene, which is the desired movement distance. The upper bound is the depth difference between the object front-facing layer and the scene. We merge all intervals for every pixel, and generate a merged depth interval set of all *non-zero* lower bounds and use the *global minimum lower* bound. See Figure 2.10 for pseudo code for merging intervals for push-to-back.

The pop-to-front algorithm is symmetric to the push-to-back algorithm. For the depth interval at each pixel, the upper bound is the depth difference between the object

back-facing layer and a front layer of the scene, which is the desired movement distance. The lower bound is the depth difference between the object front-facing layer and the scene. We merge the intervals for every pixel, and generate a merged depth interval set and use the *global minimum upper* bound.

2.11.3. Frame Buffer Encoding

We store depth values in the 1st and encode normal vectors in the 2nd and 3rd channels of the frame buffer. The 4th channel stores the pixel position where the minimum or maximum depth difference occurs. We use 32-bit floating-point frame buffers to position objects with high precision.

2.11.4. Occlusion Detection

In our system, we detect occlusion by comparing depth values in perspective depth buffers, between the closest front-facing surface of the scene and the front-facing surface of the object. If the object depth is bigger than the scene depth at every pixel, the object is completely invisible to the camera. We then call pop-to-front.

2.11.5. Depth Peeling

To support non-convex geometries, we use depth peeling [Bavoil et al., 2008] to compute all depth values for the hidden layers of the scene. During peeling we render not only depth but also normal vectors for each pixel and check for a completely blank layer to stop the iteration. We store all layer information into an array of render textures.

2.11.6. DEPTH-POP for Concave Objects in General Scenes

The version of the DEPTH-POP algorithm presented above works only for convex objects. Here we generalize it to concave objects. When the user moves a concave object, multiple depth layers of the object itself may have to be considered. The number of layers varies depending on the depth complexity of the current object. During sliding, the non-collision assumption has to be valid for all layers of the object, as the object may be sliding on any of its back-facing layers. We use depth peeling repeatedly to compute

all front and back-facing layers of the object. We then use the set of front-facing and back-facing depth layers to define the object's position and movement.

Here we discuss the generalization of the DEPTH-POP algorithm for non-convex objects and scenes. For each layer from the object and each layer of the scene, we use the depth intervals to obtain a solution for the moving distance. Each solution guarantees that a part of the concave object will be in contact with a layer of the scene, after moving the object by that distance. We gather all non-zero solutions and sort them in ascending order. We then identify the smallest moving distance that we can move the object without causing a collision. For this, we validate all potential solutions: Starting from the smallest moving distance, we examine if there would be collision if we move the object by that distance and iterate until we find a valid solution. Then we move the object by the corresponding distance. Otherwise we report failure. Figure 2.12 shows the pseudo code for the general algorithm.

2.11.7. Floating and Collision Checks

We use the set of object depth layers to detect whether the object is in contact. For each pair of depth layers of the object, we render the scene depth that is behind the front-facing object layer. We then compare the scene depth to the current corresponding back-facing object layer and compute the minimum depth difference between the depth of scene and depth of object back-facing layer. If we get a positive difference, the depth of the scene is bigger than the depth of back-facing layer at every pixel. This means that the object is floating for that layer. If the object is floating for *all* of its layers, it is not in contact with any surface. We then find the minimum depth difference across all layers and push the object back by that distance.

We detect collisions by checking for overlap between all layers of the object and the scene by iterating over the corresponding depth information. If the object is in collision (at any layer), we then find the maximum depth difference of each layer to snap the object to the front.

```
void PushToBack()
{
  // Render first layer scene depth
  SceneDepth[0] = RenderSceneDepth();
  itr = 0;

  while(1) {
    for (j=0; j<ObjectLayerNum; j++) {
      // Find all the intervals for each layer
    of the object and current layer of the scene
      PushToBackInterval(SceneDepth[itr],
      ObjectDepthFront[j], ObjectDepthBack[j]);
    }
    // Peel off current scene layer
    SceneDepth[itr+1]=DepthPeel(SceneDepth[itr]);

    // Iterations stop when next layer is blank
    if(IsBlank(SceneDepth[itr+1])) break;
  }

  // Sort all the intervals by lower bounds
  // Eliminate intervals with lower bound as 0
  SortByLowerBound (AllIntervals);

  for (i=0; i<IntervalNum; i++) {
  // Validate current solution with each layer of  the  scene  and  each
layer of the object
  ValidCount=Validate(AllIntervals[i],  SceneDepth,    ObjectDepthFront,
ObjectDepthBack);

  // If the solution doesn't cause collision
  if(ValidCount==SceneLayerNum * ObjectLayerNum){
    FinalSolution = AllIntervals[i].LowerBound;
    Break; }
  }

  if(FinalSolution > 0) {
  PushDis = FinalSolution * (CamFar - CamNear);
```

```
  // Push the object back along the mouse ray
  TranslateObject(MouseRayDirection * PushDis);

  ChangeSlidingPlane(); }
}
```

Figure 2.12. Pseudo code for push-to-back for general objects.

For simplicity, we chose to implement an image-based collision detection method, as we already compute all depth layers of the object and the scene. To accelerate computations, we carry the (new) normal vector along with the computation of the min/max depth difference. Still, our main methods are independent of the specific collision detection algorithm and other algorithms could be used, see the discussion.

Chapter 3.

Selecting and Sliding Hidden Objects in 3D Desktop Environments

In this chapter, I generalize the contact-plane sliding technique further to work with hidden objects, i.e., objects that do not meet the third assumption, assumption C of contact-plane sliding. My motivation for this generalization is to further improve 3D positioning tasks in desktop systems. I propose a new technique that allows users to select objects that are hidden from the current viewpoint without having to navigate, i.e., move the camera. I also extend the contact-plane sliding technique to work even if the manipulated object is hidden behind other objects, by making the manipulated object always fully visible through a transparency mask. The new techniques outperformed 3D widgets for selection and positioning of hidden objects in a user study. Both techniques received very positive feedback from the participants and were perceived to be fast and easy to use.

In Chapter 2, I showed that sliding with SHIFT-Sliding and DEPTH-POP is faster and more accurate than 3D widgets, for cases when the manipulated object is at least partially visible. However, this extended sliding algorithm cannot handle cases when the object is invisible. These sliding algorithms build on a visibility assumption, in two ways. First, users can only select objects that are at least partially visible, and second, the object that users manipulate must be always at least partially visible. 3D selection is typically the first task in 3D manipulation and positioning of an object typically requires that the object is selected first. To compensate for these limitations of sliding, I investigated techniques that make it possible to easily select invisible objects and subsequently to seamlessly transition to 3D positioning.

Various cues are used to judge 3D positions. Besides perspective, one of the most important cues for 3D position is occlusion [Wickens et al., 1999]. Occlusion in 3D environments could have a negative impact on some specific tasks involving discovery, access, and spatial relation of objects in 3D [Elmqvist et al., 2008]. Also, hidden objects

are difficult to select and manipulate precisely. When interacting with objects in the real world, it is challenging to manipulate them without visual cues. To select or manipulate a completely hidden object, one must rely on other cues such as tactile feedback. Similarly, in 3D user interfaces, direct selection of 3D objects is usually limited to the objects that are visible from the current viewpoint. Unless wireframe visualization in used, most interaction techniques for 3D selection and manipulation require the involved objects to be visible [Argelaguet et al., 2013]. A common work-around for selecting occluded objects is to navigate to an appropriate location so that the targets become visible. However, this navigate-to-select approach is impractical for selection-intensive applications, as performing such camera movements can take non-trivial time. If an object is, say, behind a wall, moving the viewpoint to make the object visible in the view can be time-consuming, especially in VR systems that map movement one-to-one to navigation, as the users then have to walk/navigate all the way until they are able to see behind that wall (which may be several meters away).

Another option to select hidden objects, which is frequently used in computer-aided design systems, is to use multiple views. Yet, there are still situations where an object might not be visible in any of the views, e.g., if the object is contained in another one. Then, users still need to move the camera or otherwise manipulate the view until the view is inside the outer object, which again can be time-consuming.

Various occlusion management solutions have been proposed to help users discover hidden targets. Harrison et al. [Harrison et al., 1995] proposed to use semi-transparent user interface objects in 2D interfaces. Zhai et al. [Zhai et al., 1996] surveyed the use of semi-transparency in 3D interaction. Their study showed that semi-transparency acted as an effective depth cue for 3D target acquisition. Ishak et al. [Ishak et al, 2004] used content-aware transparency in 2D graphical user interfaces, allowing users to view more content simultaneously and unambiguously. Elmqvist et al. [Elmqvist et al., 2007] proposed an image-space algorithm to achieve dynamic transparency for managing occlusion of important target objects in 3D.

Agustina et al. [Agustina et al., 2013] proposed XPointer, an X-ray telepointer technique for collaborative 3D selection, which enables users to select initially hidden objects. Their selection technique is object-based, as users can specify which of the objects intersected by the selection ray used in ray-casting should be selected. Their X-

ray manipulator is a technique for adjusting the XPointer's penetration depth. Yet, this method only works correctly in scenes with convex objects. When the selection ray hits a concave object, that entire object is made semi-transparent by XPointer. Thus, all objects within or occluded by the concave object are then visible simultaneously. Relative to revealing the scene layer by layer, this decreases the number of depth cues available for object selection. The existence of fewer depth cues also introduces an ambiguity in the selection process, which can even affect the subsequent positioning phase, as the user may not be able to perceive correctly where the manipulated object is in space.

Here, I present a new layer-based technique that enables users to select completely occluded objects, even if they are placed within non-convex parts of a scene. The "Control-Depth" selection techniques makes specific layers of the scene semi-transparent, obviating the need to move the camera. The feature involves the Ctrl-key and mouse wheel actions, without any mouse button being pressed. While users are holding the Ctrl-key pressed down, they can push the mouse wheel forward to reveal the first, previously hidden, perspective depth layer of the scene, making the initially visible surfaces in front of them semi-transparent. Every additional mouse wheel push reveals the next layer behind the previous one. If the user pulls the mouse wheel backwards, i.e., towards them, I transition the next closer semi-transparent layer back to opaque. An occluded object will then become visible after all layers in front of it are made semi-transparent. This permits the user to simply "scroll" among all visible layers. The users can then simply select the desired object by clicking "through" the transparent layers of the scene.

During object sliding, I also apply a transparency mask on the scene layers. In contact-plane sliding, if an object becomes partially invisible, it might become more difficult to position the object precisely. If an object becomes completely hidden, it is automatically brought to the front, so that the users can see it. However, this also changes the sliding plane, which might surprise novice users. According to my observations in the study in the previous chapter, some users do not have a strong expectation that the object should always be visible, which can then cause them to fail in performing a task. For my new sliding method with transparency masks, I use transparency to make the manipulated object always fully visible during positioning. When the user slides an object to a position where it is partially (or fully) hidden, the

parts of the scene in front of the manipulated object are automatically made semi-transparent. Then, the users can keep sliding the object on the desired surface. The contact-plane sliding technique with transparency masks requires the same user action as contact-plane sliding, i.e., the user slides the object with mouse movement. No extra user actions are required to enable the transparency feature.

The new sliding technique with transparency masks is independent of the "Control-Depth" selection technique. In other words, it is possible to use the new sliding technique even if object selection is limited to visible objects or if object selection is performed through some other method. In this case, the manipulated object will then still always be fully visible. Conversely, it is possible to use Control-Depth for selection and another technique for positioning other than sliding.

This chapter addresses the second research question: "**How much does layer-based transparency improve the speed/accuracy of selection and positioning of hidden objects?**" I performed a user study to evaluate the performance of Control-Depth selection as well as sliding with transparency masks. I limited my investigation to the performance of selection and precise positioning of invisible objects. I decided to compare my new technique with 3D widget-based positioning. In both conditions, the scene was shown in a 4-view display, with one perspective view and three orthogonal views. The object and target position are both invisible in the perspective view but are visible in at least one orthogonal view. With this and in the contact-plane sliding condition, the users need to use Control-Depth selection to select and to move the object through sliding. In the 3D widgets condition, and since the object is invisible in perspective, the users had to select it in one of the orthogonal views. In the following text, I refer to the condition that enables both Control-Depth selection and sliding with transparency masks as transparency sliding. I hypothesized that transparency sliding would outperform widget-based positioning for invisible objects.

I had ethics approval for the user study I performed, and participants signed consent forms. Some participants had limited experience with 3D. Since I wanted to investigate the ease of use of my techniques objectively, I did not include experts in my study. Experts can have strong opinions on usability, based on their "favourite" 3D manipulation systems. My solutions deviate from the predominant VR/desktop paradigms, which is why I chose to evaluate my solution with users that exhibit less bias

in this respect. The results from the user study confirmed the hypothesis. Transparency sliding was overall significantly faster than 3D widgets in terms of completion time (selection & positioning time). Looking only at the time users took for the selection, it was approximately a fifth of the positioning time. In terms of error distance, there was no significant difference between transparency sliding and 3D widgets. Ten out of twelve participants preferred transparency sliding over 3D widgets.

Transparency sliding was rated positively. Some of the participants even asked for the semi-transparency feature in the 3D widgets condition, as the object was usually hidden in the perspective view during object movement in that condition. I see this as motivation to include the semi-transparent features in standard 3D software. Although I had novice and intermediate participants in the user studies, I believe that my techniques will benefit users with all levels of 3D expertise. The new features would help 3D designers improve their speed during 3D modelling, especially when dealing with hidden objects. The details about the two techniques, experimental design, user studies, and results can be found in the following published paper.

I exploit the computing power of GPUs and use the frame buffer for the computations. To render textures as semi-transparent, the alpha value of textures in the scene are automatically changed depending on the depth values of the manipulated object. If the manipulated object is farther away from the camera than an object in the scene, each occluder is shown semi-transparent at pixels on or around the manipulated object. For the layer-based Control-Depth selection, I use depth peeling [Everitt, 2001] to identify the visible layers of the scene. The techniques work even with concave scenes. The detail of the implementation can be found in the following published paper.

This work was previously published in a full paper titled “Selecting and Sliding Hidden Objects in 3D Desktop Environments” at the Graphics Interface conference in 2019. I co-authored this paper with Wolfgang Stuerzlinger.

Abstract

Selecting and positioning objects in 3D space are fundamental tasks in 3D user interfaces. We present two new techniques to improve 3D selection and positioning. We first augment 3D user interfaces with a new technique that enables users to select

objects that are hidden from the current viewpoint. This layer-based technique for selecting hidden objects works for arbitrary objects and scenes. We then also extend a mouse-based sliding technique to work even if the manipulated object is hidden behind other objects, by making the manipulated object always fully visible through a transparency mask during drag-and-drop positioning. Our user study shows that with the new techniques, users can easily select hidden objects and that sliding with transparency performs faster than the common 3D widgets technique.

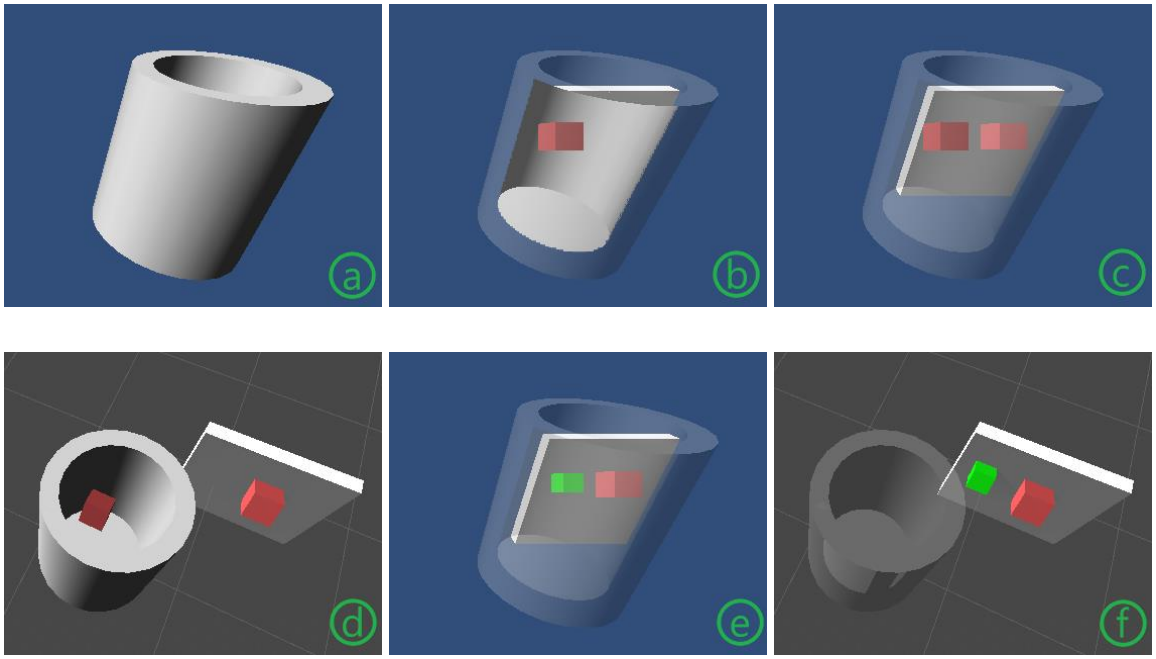


Figure 3.1. **(a) A cup with standard opaque rendering. (b) The first layer rendered semi-transparent, revealing the red cube inside the cup. (c) The first two layers (the entire cup) rendered semi-transparent, revealing a second red cube behind the cup. (d) Top view corresponding to object positions in (a), (b), and (c). (e) The cube inside the cup moved to the surface behind the cup, highlighted in green. (f) Top view corresponding to object positions in (e).**

Keywords

3D positioning; selection; 3D interaction; transparency.

Index Terms

H.5.2. Information interfaces and presentation (e.g., HCI): User Interfaces - Graphical user interfaces (GUI)

3.1. Introduction

Selecting and posing a 3D rigid object, i.e., manipulating the position and orientation of an object, is a basic task in 3D user interfaces. Such tasks can be time-consuming, because 3D selection can require control of 3 degrees of freedom (DOFs), and full manipulation can involve 6 DOFs: Three DOFs for translation along three axes and another three for rotation around three axes.

In a virtual environment, the direct selection of 3D objects limits the user to the objects that are within their reach. Unless wireframe visualization is used, selection techniques typically limit selection to all visible objects [Argelaguet et al., 2013], which requires only 2D input. In such systems, the only option to select hidden objects is to move the camera so that the desired object becomes visible. Another option, which is frequently used in computer-aided design systems, is to use multiple views, but (with hidden surfaces) even then there are situations where an object might not be visible in any of the views, e.g., if the object is contained in another one. Here, we present a new layer-based technique that enables users to select any occluded object, even if it is placed within non-convex parts of a scene.

Many manipulation techniques rely on 3 or 6 DOF input devices, based on a one-to-one mapping of input and object movement. Research has shown that 3DOF input devices outperform 2D devices in some contexts. Yet, people are more familiar with the form and function of a mouse [Bowman et al., 2004].

For 3D manipulation of objects with 2D devices, it is challenging to provide efficient mappings between the 2D input and the 3D object movement. However, there is evidence that 2D input devices can outperform 3D devices for certain 3D manipulation tasks, through a smart mapping of user input to intuitive 3D object movement [Bérard et

al., 2009]. One such approach is constraint-based positioning, where the position of the manipulated object is constrained by the attributes of other objects of the scene.

Sliding is a prime example of a constraint-based 3D positioning technique [Oh et al., 2005]. Here, the object follows the mouse cursor and slides on the surfaces behind it. The object movement is then defined by the surface the object is in contact with. More advanced sliding methods perform better than the commonly used 3D widgets [Strauss et al., 1992], even for objects that are not in contact with other surfaces. Yet, sliding assumes that the manipulated object is always (at least partially) visible. If an object becomes hidden, it is brought to the front, so that the users can see it. Here we address this limitation of the original sliding method through semi-transparency, by making the manipulated object always visible, even if the object is behind other ones.

We performed a user study to evaluate our new selection and sliding techniques. For this we used a task that requires the user to select a 3D object (initially) hidden in the main perspective view and position it into an (initially) hidden target position. We hypothesized that both our new layer-based selection and transparent sliding technique would perform better than 3D widgets.

Below, we first review relevant object selection and manipulation methods. Then we discuss original sliding, our new extension and describe our user study. Finally, we discuss the results and mention potential future work.

3.1.1. Contributions

The main contributions we present here are:

- A new layer-based technique to select hidden objects in non-convex scenes. This technique reveals the scene layer by layer, which enables the user to select an object on any desired scene layer.
- A new technique to show the manipulated object during sliding, regardless if it is directly visible or behind other parts of the scene.

3.2. Related Work

There has been substantial research in the field of 3D manipulation. Overall, the choice of the "best" input device depends on the task and the hardware platform. Some 3/6DOF devices perform better than the mouse on specific tasks, e.g., the Control Action Table [Hachet et al., 2003], the GlobeFish, and GlobeMouse [Froehlich et al., 2006]. However, the mouse is generally more efficient than 3/6DOF devices for accurate placement, despite the lack of a third DOF [Bérard et al., 2009] [Sun et al., 2018].

Mouse-based 3D selection and manipulation is not without its limitations. First, one can only select visible objects with a mouse. Second, simultaneous translations along all three directions are not possible, due to the 2D nature of the device. One way to compensate is through smart mappings that use constraints, either explicitly specified by the user in their interaction, or implicitly specified through the content of the scene. Third, 3D rotations are not efficiently supported. Most 2D based interaction techniques limit the rotations to one axis (or at most two axes) at a given time.

Transparency has been widely used in 2D and 3D user interfaces. Zhai et al. [Zhai et al., 1996] surveyed the use of semi-transparency in 3D interaction. Their study showed that semi-transparency acted as an effective depth cue for 3D target acquisition. Bier et al. [Bier et al., 1993] introduced a 2D transparent lens. Harrison et al. [Harrison et al., 1995] proposed to use semi-transparent user interface objects in 2D interfaces. Harrison et al. [Harrison et al., 1995] studied how to minimize interference between transparent layers. Gutwin et al. [Gutwin et al., 2003] studied the effects of dynamic transparency on targeting performance. Ishak et al. [Ishak et al., 2004] introduced a content-aware transparency mechanism that dynamically adapts opacity depending on the importance of various parts of a window.

There has been substantial research on employing transparency in the visualization of 3D scenes, often to support navigation. Chittaro et al. [Chittaro et al., 2001] studied if semi-transparency is useful for 3D navigation in virtual environments. They found some positive effects on user navigation performance. Diepstraten et al. [Diepstraten et al., 2002] introduced a view-dependent transparency model. Coffin et al. [Coffin et al., 2006] presented cut-away techniques that permit a user to look "through" occluding objects, by interactively cutting holes into the occluding geometry. Elmqvist et

al. [Elmqvist et al., 2007] proposed an image-space algorithm to achieve dynamic transparency for managing occlusion of important target objects in 3D. Their techniques yielded significantly more efficient performance in 3D navigation tasks.

Various techniques have been proposed to improve the visualization of overlapping objects. LayerFish [Webb et al., 2016], supported layering and manipulating overlapping content in a 2D design space on desktop surfaces. To reduce demands on effort and attention, it used the fisheye technique to render an in-place scene index. Ramos et al. [Ramos et al., 2006] presented two techniques for 2D layering operations. The first provided a graphical representation of a cascaded stack of layers above the selected elements. The second used a 'splatter' effect to radially distribute overlapping elements. Davidson et al. [Davidson et al., 2008] proposed a depth sorting technique that extended standard 2D manipulation techniques and combined the layering operation with a page-folding metaphor for more fluid interaction in applications requiring 2D sorting and layout. Luboschik et al. [Luboschik et al., 2010] proposed a weaving technique, which offered a new and effective alternative for picking any object from a set of overlapping objects, without using transparency. Javed et al. [Javed et al., 2011] presented a novel approach to manage occlusion between physical items resting on tabletop displays and virtual objects projected on the display.

Agustina et al. proposed XPointer [Agustina et al., 2013], an X-ray telepointer technique for collaborative 3D selection, which enables users to select initially hidden objects. Their selection technique is object-based, as users can specify which of the objects intersected by ray-casting should be selected. The 3D editing operations for XPointer are standard widget-based editing techniques, inherited from Autodesk Maya. Their X-ray manipulator [Agustina et al., 2013] is a technique for adjusting the XPointer's penetration depth, but this method only works correctly in scenes with convex objects. When the selection ray hits a concave object, that entire object is made semi-transparent by XPointer. Thus, all objects within or occluded by the concave object are then visible simultaneously. Relative to revealing the scene layer by layer, this decreases the number of depth cues available for object selection (and later manipulation). The existence of fewer depth cues also introduces an ambiguity in the selection process, which can even affect the subsequent positioning phase, as the user may not be able to perceive correctly where the manipulated object is in space.

Oh et al. [Oh et al., 2005] presented a sliding algorithm, where the object follows the cursor position directly and slides on any surface behind it, i.e., the moving object always stays attached to other objects. Sun et al. [Sun et al., 2016] improved Oh et al.'s work through SHIFT-Sliding and DEPTH-POP, two techniques that significantly speed up common 3D positioning tasks, including tasks that require the object to float in mid-air. However, all such sliding algorithms build on a visibility assumption in two ways. First, users can only select objects that are at least partially visible, and second the object that the user manipulates must always be at least partially visible to the user. If an object becomes hidden, these sliding techniques (have to) bring the object to the front, so that users can see it. Similarly, DEPTH-POP enables the user to place the object along any position along the mouse ray, as long as it is at least partially visible.

3.3. Selecting and Sliding Hidden Objects

Here, we first present our new method to select hidden objects. Then we detail our new technique to facilitate the sliding of objects even if they are hidden.

3.3.1. Control-Depth Selection

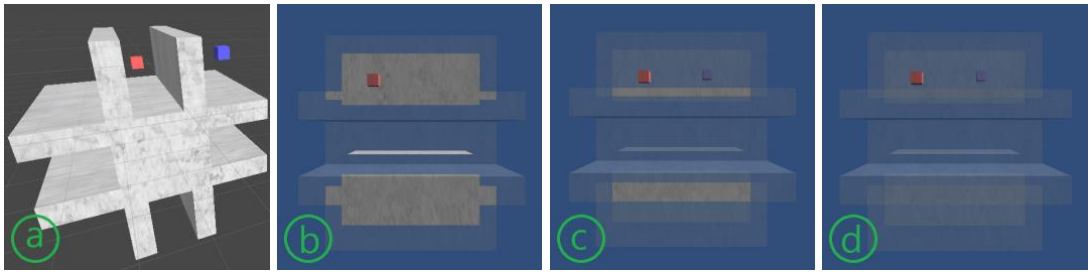


Figure 3.2. **(a) A concave object with other objects in the cavities. (b) and (c) Seen from the side, the red and blue cubes are behind different layers of the object. Here, layer-based selection can reveal the desired object by making all content in front of it semi-transparent. (d) Object-based transparency makes the whole object transparent, which makes it harder to understand where the cubes are.**

Selection from a specific layer of the scene is necessary in many situations. Figure 3.2 left shows a scene where the red and blue cubes are positioned behind

different layers in the scene. Layer-based selection is useful when there is a need to select an object behind a specific layer.

With all 2D input-based interaction methods that rely on a single view, users can only select visible objects. Hidden objects cannot be selected without moving the camera or transitioning to other view types, such as wireframe. In Figure 3.4 and with such techniques, if there is an object at position C, D or E in the scene, the object cannot be selected without camera navigation. To enable the user to select such a hidden object, we introduce our new “Control-Depth” selection technique, which involves the Ctrl-key and mouse wheel actions. We use the Ctrl-key to activate the depth selection mode. While users are holding the Ctrl-key pressed down (without having clicked any mouse buttons), they can push the mouse wheel forward to reveal the first, previously hidden, perspective depth layer of the scene, making the initially visible surfaces in front of them semi-transparent. Every additional mouse wheel push reveals the next layer behind the previous one. If the user pulls the mouse wheel backwards, i.e., towards them, we transition the next closest semi-transparent layer back to opaque. An occluded object will then become visible after all layers in front of it are made semi-transparent. This permits the user to simply “scroll” among all visible layers. The users can then simply select the desired object by clicking “through” the transparent layers of the scene. In comparison to XPointer [Agustina et al., 2013], our new technique enables us to better deal with scenarios with concave scene objects. We use depth peeling [Everitt et al., 2001] to identify the visible layers for Control-Depth selection.

Figure 3.3 and 3.4 show the same 3D scene with a concave object, with Figure 3.4 showing a side view. The position of the red cube in Figure 3.3 corresponds to position D in Figure 3.4. XPointer would make the whole base object semi-transparent, as illustrated in Figure 3.3(d), which makes it difficult to judge the position of the red cube, unless the user has the strong prior knowledge of the scene. Figure 3.1 shows an example where XPointer’s design decision to make whole objects transparent introduces some ambiguity as to where objects are located. The first red cube is inside the cup and the other one is behind the cup. Their sizes are similar in the perspective view. If the whole cup is made semi-transparent, it is hard for the average user to distinguish which of the two red cubes is inside the cup.

The idea of using the mouse wheel to select objects that are behind other objects has previously been presented in COMSOL Multiphysics[1], which relies on a rendering mode that shows *all* objects *simultaneously* as semi-transparent. Yet, research into volume rendering methods, which are all based on semi-transparency, has shown that even the best volume rendering methods result in 25% error in terms of depth perception [Englund et al., 2016]. Standard volume rendering, which renders everything semi-transparent from back to front, performs even worse. Based on this result, we decided to pursue an approach that renders everything in front of the layer that the user is currently focusing on semi-transparent, and everything behind that opaque, as this makes it easier to understand the geometric context. Moreover, we enable the user to directly control which layer is shown, which makes it easier to understand the geometric relationships inside the scene.

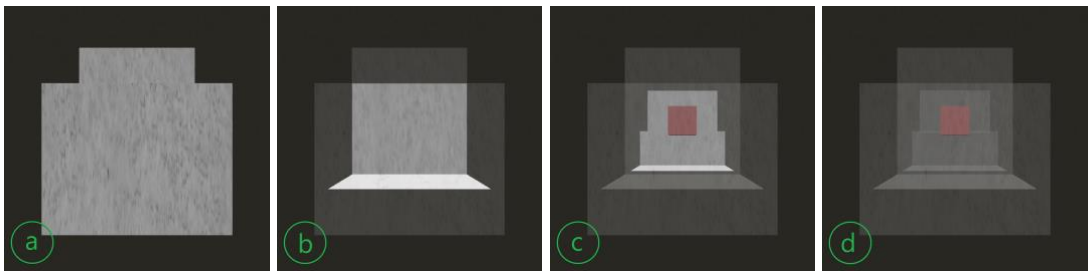


Figure 3.3. **(a) A scene with standard opaque rendering. (b) The first layer rendered semi-transparent. (c) The first two layers rendered semi-transparent, revealing the red cube. (d) The entire object rendered semi-transparent, which corresponds to the visualization used by XPointer. It is hard to see if the cube is in front of the small ledge or behind it. In reality, the cube is in front of the ledge at position D, see Figure 3.4.**

Figure 3.3 illustrates the process of Control-Depth selection. Figure 3.3(a) shows the original view of the scene. In Figure 3.3(b), the first layer of the scene was made semi-transparent by moving the mouse wheel forward once, while holding the Ctrl-key down. In Figure 3.3(c), the second layer was made semi-transparent, revealing the red cube. The user can then simply select that cube by moving the mouse cursor over it and

[1] https://www.comsol.com/

selecting it through a click. For selection, we select the first non-transparent object below the cursor.

3.3.2. Transparency Sliding

We base our design for our new transparency sliding method on a basic sliding algorithm [Oh et al., 2005]. By enhancing sliding through semi-transparency, we make it possible to keep objects visible during sliding. In basic sliding, the manipulated object moves along the surface behind it that it is in contact with. During such sliding, the object is always (at least partially) visible to the user. Users can then only select and manipulate visible objects, such as objects at A or B in Figure 3.4. They could not move an object to position C, D or E, as it would be fully occluded.

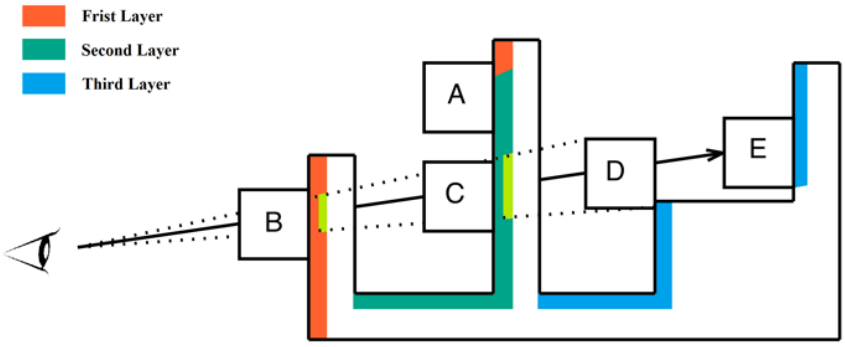


Figure 3.4. Illustration of Control-Depth selection and transparency sliding. Objects C, D, and E are fully occluded, but can be revealed with Control-Depth selection (and then selected). The first, second, and third (front-facing) layer of the concave object is shown in orange, green, and blue, respectively. All layers can be viewed by transparency sliding, which makes surfaces before the object semi-transparent, e.g., the two areas in light green when the object is sliding at position D.

For our new transparency sliding method, we use transparency to make the manipulated object always fully visible during manipulation. When the user slides an object to a position where it is partially (or fully) hidden, we make the parts of the scene in front of the manipulated object semi-transparent. In Figure 3.4, the object is fully

occluded at position C, D, or E. To enable the user to manipulate the object at these positions, we make the parts that are occluding the object semi-transparent. Thus, the two areas in light green are made semi-transparent so the users see and manipulate the whole object when it is at position D. With DEPTH-POP [Sun et al., 2016], users can then place the object at positions B, C, D, or E, while still being able to see it due to the transparency mask (for C, D, and E).

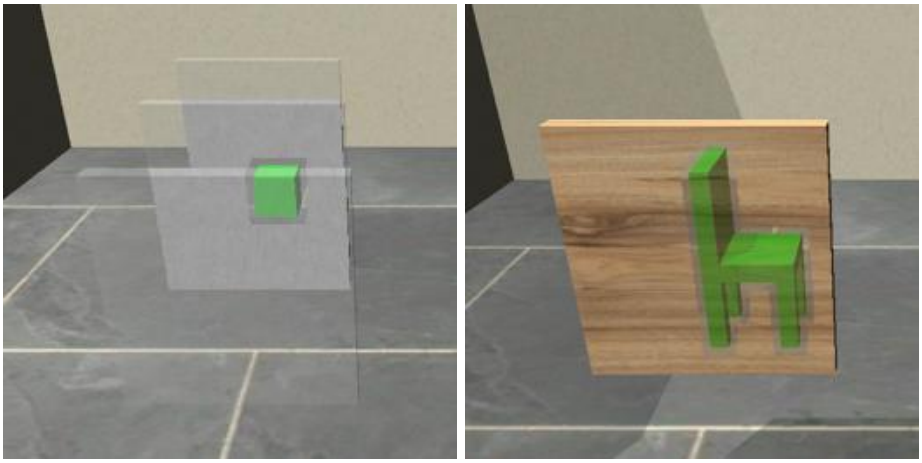


Figure 3.5. Transparency masks around a manipulated object.

Figure 3.5 shows the transparency masks around a manipulated object. With it, the object is always fully visible during sliding. Figure 3.5 left shows a combination of Control-Depth transparency and transparency mask. Through Control-Depth transparency, the first layer of the scene is made semi-transparent. If users slide the cube behind the second and third layer of the scene, the scene in front of the object is made semi-transparent with the transparency masks.

3.3.3. Implementation

We exploit the computing power of GPUs and use the frame buffer for the computations. The alpha value of textures in the scene depends on the depth values of the manipulated object. If the manipulated object is farther away from the camera than an object in the scene, each occluder is shown semi-transparent at pixels on or around the manipulated object. The border size around the object (in screen space) is predefined but can be easily adapted to different scenarios. We set the border size to 3 pixels in our experiments.

To implement our new transparency mask, we use an algorithm that is an extension of the one used for Control-Depth selection. We apply a surface shader on all objects in the scene. We pass the current scene depth texture and the depth texture of the manipulated object (if there is one) to the surface shader. With Control-Depth selection, we update the current scene depth texture with depth-peeling. Initially, the current scene depth texture is defined from the first layer of the scene, which represents the current visible layer. If the user pushes/pulls the mouse wheel forward/backward, the scene depth texture is set to the next, respectively previous layer. In the surface shader, when the depth value of the pixel on the current object is less than the current depth texture (i.e., closer to camera), that pixel's alpha value is set to a constant, currently 0.25. This is the core of the technical implementation of Control-Depth selection. To implement the transparency mask, we set the alpha values of a pixel that are closer to the camera than the manipulated object to 0.25 and also spread that transparency value to the 3x3 surrounding pixels.

To enable the transition between different visibility layers during sliding, we use a generalized version of the DEPTH-POP algorithm [Sun et al., 2016], which does not check if the object remains visible. As shown in Figure 3.5, the cube in the left image and the chair at the right are both fully hidden yet made visible by transparency masks. Users can use the generalized DEPTH-POP technique to transition between different layers.

3.3.4. Combining Selection and Sliding

During prototyping, we identified that it is useful to enable Control-Depth even during sliding. Once users have selected an object, if they hold the Ctrl-key and left mouse button down at the same time, we perform both Control-Depth and DEPTH-POP operations simultaneously, which reveals depth layers while positioning the object between them. However, note that even with this combination, the manipulated object can become partially hidden behind parts of the scene. In other words, even with the layer-based method there are situations when the transparency mask is needed to show the manipulated object and the context around it correctly.

Finally, we point out that our new transparency sliding technique is independent of the selection technique. In other words, it is possible to use the new sliding technique even if object selection is limited to visible objects or if object selection is performed

through some other method. In this case, the manipulated object will then still always be visible. Conversely, it is possible to use only the new transparency sliding technique without enabling the selection of hidden objects.

3.4. Users Study

We performed a user study to evaluate the performance of Control-Depth selection as well as transparency sliding. We initially considered a comparison with the X-Ray manipulator in XPointer. Yet, we chose to not to do this, as the XPointer technique [Agustina et al., 2013] cannot handle the positioning of objects in scenes with concave parts appropriately, as discussed above. Also, XPointer requires a side view visualization for the cursor, which increases the time spent in the selection phase, as the user must move the cursor over larger distances (between windows). Moreover, Sun et al. [Sun et al., 2016] had already shown that sliding with DEPTH-POP is more efficient and accurate than 3D widgets, for cases when the manipulated object is at least partially visible. Still, DEPTH-POP cannot handle cases when the object is hidden.

We also considered a comparison between basic sliding with camera navigation and transparent sliding. When we performed a pilot study with novices, camera navigation turned out to be a major challenge, as they had no experience with 3D editing systems. While they could move the camera to a position where they could see the object, they then struggled to (slowly) find a viewpoint that shows the target position in the same view, which is the only option that enables users to perform the task with basic sliding. As our interaction techniques are targeted at novices, we decided to disable camera navigation in our experiment, as navigation would dominate the timings and thus pose a confound.

Based on these arguments, we limited our investigation to the performance of selection and precise positioning for hidden objects. Given that the XPointer technique and most 3D packages use 3D widgets for manipulation, we decided to compare our new technique with 3D widget-based manipulation. As our new user interface is designed to make the task of both selecting and manipulating hidden objects faster, we hypothesized that we would get similar results as Sun et al. [Sun et al., 2016], i.e., transparency sliding would outperform 3D widget-based manipulation for hidden objects.

3.4.1. Participants

We recruited 12 (7 female) unpaid undergrad students from the local university population. We did not screen participants for 3D experience. Our participants had varying game expertise, with 42% playing games regularly.

3.4.2. Apparatus

We built our system in the Unity game engine. We used a desktop computer with 3.5 GHz i7 processor, 16 GB of memory, and two NVIDIA GeForce GTX 560 SLI graphics cards. We used a mouse and a keyboard as input devices.

3.4.3. Procedure

We designed a 3D object positioning experiment and asked participants to move an object from a hidden start position to a hidden target position in various scenes. We used two different levels of depth complexity for the trials. All target positions were in contact with other surfaces in the scene. In all conditions, the scene was shown in a 4-view display, with one perspective view and three orthogonal views. Figure 3.6 shows two sample tasks in the sliding condition. The object and target position are both hidden in the perspective view, yet the source and target object locations are marked with a red and blue marker respectively. The object is the red cube, whereas the target position is rendered as a semi-transparent blue cube, a 3D copy of the object. The horizontal distance on screen between object and target positions are roughly one half of screen width in the perspective view. With this and in the sliding condition, the users need to use Control-Depth selection to select and to move the object through sliding in the perspective view.

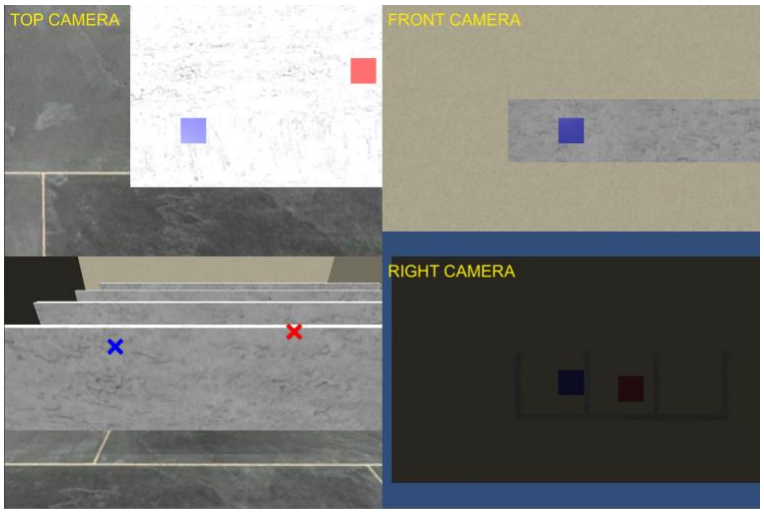
The geometry of the “building” in Figure 3.6 (b) consists of walls and the floor as a single, large, non-convex object. In this task scenario, the XPointer technique [Agustina et al., 2013] makes the entire non-convex object transparent, which then makes it challenging (or impossible) to position an object at a location hidden by a wall onto the floor (or even another wall) behind that first wall. Our layer-based selection method works properly in this scene. In the 3D widgets condition, and since the object is

hidden in perspective, the users had to select it in one of the orthogonal views. After the users click on the object, the 3D widgets appear on the object, and the 3D widgets can be used in all four views for manipulation. Having to use different views typically slows the selection process down, yet the only other option to select a hidden object would be to permit camera navigation, something that we wanted to avoid due to the results of our pilot study mentioned above.

There was a 2-minute training session before each condition, which introduced participants to the techniques in a playground environment but did not include any version of the experimental tasks. We asked the participants to perform the tasks as quickly and as accurately as possible. After the participants finished all the tasks, we asked them to fill a questionnaire about the usability of the two techniques. We also asked them about potential improvements. In total, the study took about 30 minutes for each participant.

3.4.4. Experimental Design

The experiment compared transparency sliding and 3D widget-based positioning in a within-subjects design. Technique was the only independent variable. When the user had positioned the object at the target position, they needed to press the space bar for confirmation, and then went on to the next trial. The users were allowed to use their non-dominant hand to press the space bar. Each participant performed 20 trials, ten trials for sliding and ten for 3D widgets.



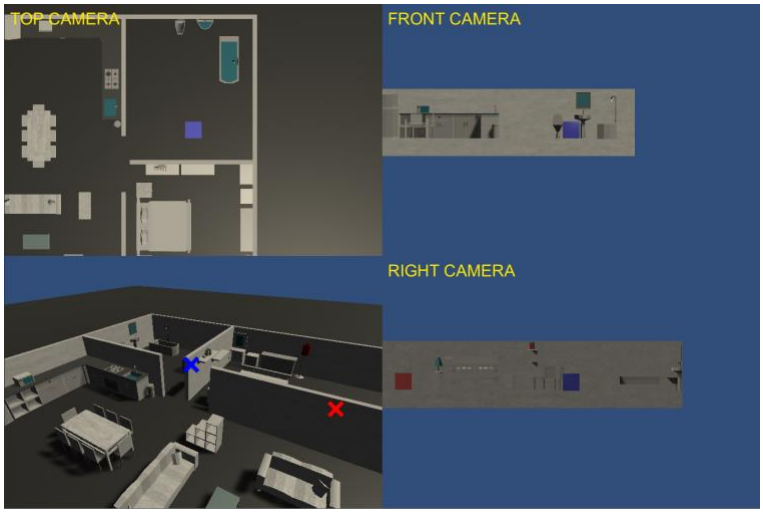


Figure 3.6. **Two sample tasks in the sliding condition a) task scene with simple geometry and b) a task scene with richer geometry. The object and target positions are both hidden in the perspective view. They are marked by red and blue markers respectively. The target position is centered in the three orthogonal views. The object (red cube) is visible in at least one orthogonal view.**

3.4.5. Data Generation

We measured the combined “visual search & selection” time, the completion time, and relative error distance from the ideal target position. Time was measured in seconds. For the “visual search & selection” time, the timing starts when the previous trial ends, and ends when the correct object is selected. For the completion time, the timing starts when a mouse button or a key is pressed, and the timing includes both the selection and positioning stages. The timer for completion time ends when the user releases the mouse button for the last time. The error measure was calculated as the absolute distance to the target (center to center) over the object size (length of any side of the cube). We recorded all actions of each user.

3.4.6. Results

We performed paired t-tests to compare the mean in “visual search & selection” time, the completion time, and relative error distance from the ideal target position for sliding and 3D widgets.

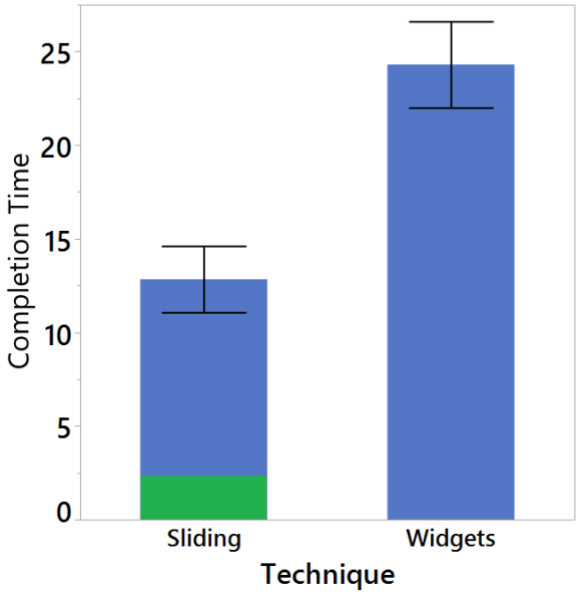


Figure 3.7. Average completion times (in seconds) for two techniques. Each error bar is constructed using a 95% confidence interval of the mean. The green area in the sliding condition represents the selection time of 2.31 seconds, whereas the corresponding time with 3D widgets is 0.

The average time for “visual search & selection” with the 3D widget-based method was 3.07 seconds, while Control-Depth selection took 4.65 s. Looking only at the time users took for the selection (i.e., the time from pressing the Ctrl key until the correct object was selected, which includes the mouse movement to select the object) in the Control-Depth condition, it took users only 2.31 s.

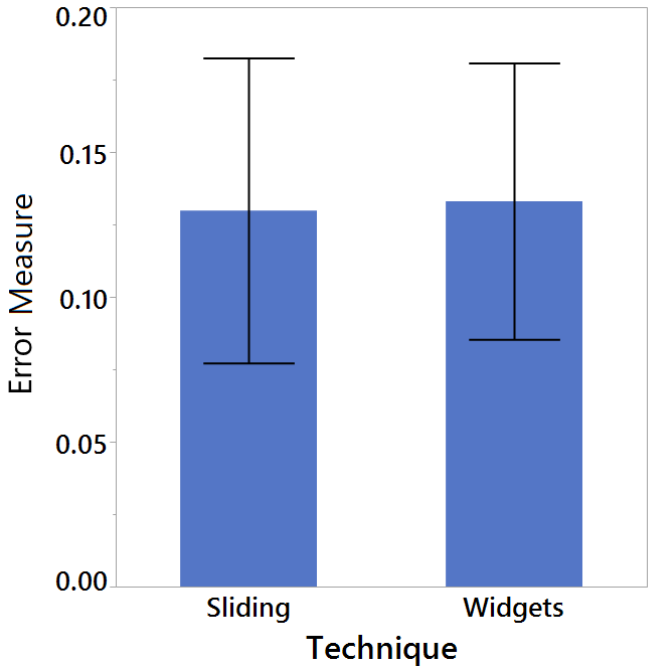


Figure 3.8. Average error measures for two techniques. Each error bar is constructed using a 95% confidence interval of the mean.

We call the total of the selection and positioning time the completion time. This timer starts when the user pressed the mouse/keyboard for the first time. The results showed that sliding (M = 12.84s, SD = 9.82) is overall significantly faster than 3D widgets (M = 24.31s, SD = 12.74), $t(119) = 8.3192$, $p < .0001$ in terms of completion time. For the sliding condition, the above-mentioned average selection time of 2.31 seconds represents approximately a fifth of the positioning time. The selection time for the 3D widgets condition is zero, as timing started with the first mouse click. See Figure 3.7.

In terms of error measure, there was no significant difference between sliding (M = 0.130, SD = 0.291) and 3D widgets (M = 0.133, SD = 0.264), $t(119) = 0.09$, $p = 0.93$. See Figure 3.8.

Almost all, 11 out of 12, participants thought sliding was easy to use, while 10 participants thought 3D widgets were easy to use. Still, 10 out of 12 participants preferred sliding over 3D widgets.

3.5. Discussion

We first compared only the “visual search & selection” time. The results showed that click-based selection used in the 3D widgets condition is 34% faster than Control-Depth selection. This is not surprising, as Control-Depth selection requires more actions from the users. However, the difference in time is only 1.58 s. According to Brown et al.’s results [Brown et al., 2014], an average mouse movement takes about 0.8 s. Therefore, we can approximate the visual search time as (3.07 - 0.8 =) 2.27 s for click-based selection, and (4.65 - 2.31 =) 2.34 s for Control-Depth selection, which are unlikely to be significantly different.

Although selection takes slightly more time with the Control-Depth technique, the results in terms of the total completion time support our hypothesis. Even though the timing for 3D widgets did not include the mouse movement time before selection, sliding was still faster than 3D widgets, even for hidden objects. Users’ prior knowledge of the task scenes might have an impact on the performance, but none of the participants seemed to have been familiar with the scenes. Naturally, if users know which layer the object is on beforehand, they could perform better in the tasks. As we targeted our interaction design at novices, we only recruited novice users unfamiliar with 3D editing software. Obviously, experienced users might be able to complete the investigated tasks with a combination of basic sliding movements or 3D widget manipulation together with camera navigation. Yet, we believe that experienced users will still benefit from our new technique, as their stronger mental model of the geometry of a scene will help them know/remember where an object is located, even if it is not visible. This knowledge, together with Control-Depth selection, will permit them to avoid camera navigation altogether in many instances, which will make their workflow more efficient.

In our experiment, users seem to have had little issues with understanding the scene geometry in the sliding condition. They showed no signs of confusion with the Control-Depth selection, transparency masks, or their combination, even though they both introduced transparency to the scene. The two techniques did not conflict with each other in any of the task scenes, e.g., Figure 3.5 left. Additionally, the border of the transparency mask provided cues for users to understand which layer the manipulated object is at. It also helps that the Control-Depth selection is associated with discrete

actions in the selection stage, while the transparency mask involves a continuous sliding action during the manipulation stage. From our observations during the experiment, we can also confirm that the techniques are easy to understand and learn and all users were able to use them after a short training session.

Most participants preferred sliding over 3D widgets in our task scenarios, which involved situations where the object is hidden at both the source and target locations. Some participants identified correctly that sliding required less mouse movement and thus rated it more positively. Also, with sliding, the object was always under the mouse cursor and fully visible, which led the participants to feel they had a better control over the object's position. Some of them even asked for the semi-transparency feature in the 3D widgets condition, as the object was usually hidden in the perspective view during object movement in that condition. We see this as motivation to include our semi-transparent techniques in standard 3D software. This would help 3D designers improve their efficiency during 3D modelling.

We use mouse wheel operations together with the Control modifier key (Ctrl). This is very similar to how 3D packages use mouse operations together with modifier keys, e.g., to move objects in three dimensions. After being exposed to our new technique once, all participants had no difficulties using it during the study. Depending on the application scenario, other activation methods can be used, including through side buttons on a mouse (or controller) instead of the Ctrl key, which then also obviates the need for a keyboard. Mappings for touch screens, e.g., through the use of two-finger gestures, are equally possible.

In the basic sliding technique, an object is automatically popped to the front whenever occluded. Novice users might find this behaviour unexpected. The new semi-transparent sliding technique presented here does not share this automatic behaviour, which gives the user better control during interaction. If the users want to pop an object to a layer in front, they can explicitly use DEPTH-POP during sliding. Our participants also commented positively on how the new interaction technique handles occlusion.

Unlike object-based selection techniques, our layer-based technique allows users to select an object from a specific layer. This avoids any potential ambiguity in perception of object depth, especially when there are concave objects in the scene. With

our techniques, users can perform object selection and manipulation continuously within a single perspective view. Any object (visible or not) within the view frustum can be selected and positioned to any desired position that meets the sliding assumptions [Sun et al., 2016], without having to change the camera view. This makes our new techniques very general.

The transparency mask technique could trivially be combined with SHIFT-Sliding [Sun et al., 2016]. When the user lifts an object up into a floating state, i.e., without any contact to other surfaces, and then starts sliding parallel to the original contact plane, we could use transparency masks to make the manipulated object fully visible if the object becomes partially hidden. This generalizes transparency sliding to floating objects.

We used depth buffers to implement the Control-Depth selection method. We reveal one scene layer at a time with a mouse wheel action. Most computations are performed in the graphics hardware and for the scenes used in the experiment, it takes approximately at most an extra 4 ms to reveal each layer. This temporarily caused the frame rate to drop from 69 fps (14.5 ms per frame) to 55 fps (18.2 ms per frame). During sliding with transparency masks, the frame rate never dropped below 69 fps. During DEPTH-POP, the frame rate dropped to as low as 47 fps (21.2 ms per frame), as it requires rendering more object/scene layers. Users did not seem to be affected by the small latency introduced by the techniques.

3.6. Limitations

Our technique is efficient, as users can quickly identify, select and slide the desired object. However, we anticipate that on lower-end graphics hardware the current implementation of Control-Depth selection could slow down in scenes with high depth complexity, as we focused on the interaction and did not realize all potential optimizations in the code. This can be addressed with a redesign of the computations.

One implementation problem we face is z-fighting, where two polygons have similar values in the z-buffer at the same pixel. In Control-Depth selection, we constantly compare the object depth with the scene depth. Insufficient precision in the depth buffer then introduces z-fighting, which makes it hard to identify layers. One option to reduce this problem is to adjust the length of the camera frustum for each specific scene.

For Control-Depth selection and transparency mask, we use a fixed alpha value with the standard transparent rendering approach to achieve a semi-transparent effect, where the target/manipulated object is shown with a blended colour. This effect generates different results for objects with different textures. Moreover, image quality potentially degrades with an increasing number of depth layers in the scene, which might make it more difficult for users to judge the manipulated object's position. We also considered using cutaways. Yet, using such visualization techniques would cause the object to slide on "ghost" surfaces, i.e., would involve sliding on surfaces that are then completely hidden, which likely would surprise novices. Finally, our current work focused on the interaction aspects. Thus, we consider optimized semi-transparency effects that maximize the visibility of one or more objects to be out of scope for our current work, but plan to revisit this in the future.

3.7. Conclusion and Future Work

For applications where users are positioning objects with a mouse in a 3D scene, we presented our new Control-Depth selection method, which enables users to select hidden objects in arbitrary scenes without resorting to (potentially time-consuming) camera navigation, by iteratively "peeling away" layers in front of a target object and then simply selecting it. We also proposed a new extension to the sliding technique, which uses a transparency mask to facilitate object sliding, so that the user can see the object even if it slides behind other objects.

We performed a user study to compare the performance of our new selection and transparency sliding techniques with common 3D manipulation widgets. Users had to select hidden objects and move and precisely position them at hidden target positions. The results showed that even though it took an extra step to complete the task in the sliding condition (selection and positioning), the combined time was still significantly faster than with common 3D widgets. Users found our techniques easy to learn and use.

In the future, we plan to investigate methods to optimize our implementation of Control-Depth selection and reduce the impact of z-fighting. We may also investigate how to optimize rendering of semi-transparency effects for different textures and

geometries. Additionally, we will check if more complex scene geometry affects the users' perception of the scene when using our techniques.

In order to further evaluate the efficiency and simplicity of the techniques, we would also like to perform another experiment with both novices and participants that are familiar with desktop-based 3D interaction. There we would evaluate how much benefits our techniques have for experts.

Chapter 4.

Comparing Input Methods and Cursors for 3D Positioning with Head-Mounted Displays

In this chapter, I start investigating 3D positioning in an immersive environment. I apply contact-plane sliding in a virtual reality (VR) system with a head-mounted display (HMD) and compare the performance of different input devices and cursor displays. My motivation is to examine if sliding works well in VR and if so, which input device and cursor display is the fastest and most accurate method for 3D object sliding. The mouse outperformed the 3DOF controller in a 3D positioning task. Stereo cursor and one-eyed cursor did not demonstrate a significant difference in speed or accuracy.

VR headsets have become prevalent in recent years and are starting to be widely used with computer games. More game developers than ever are helping with the VR revolution in the game industry. Also, 3D modelling applications in VR can provide 3D designers an immersive environment, which could inspire them in a way that desktop systems cannot. Designers can then be “inside” the 3D world they are creating. The HTC Vive was unveiled in 2016 and has become a standard in the industry and includes two wireless handheld controllers. Hundreds of games and applications are now available for the HTC Vive. For both games and VR applications, 3D positioning is a fundamental task. Therefore, my next step is to explore 3D positioning in VR.

Input devices are an important component in designing, developing, and using 3D user interfaces. There are many different types of input devices to choose from and some of them may be more appropriate for certain tasks than others. Despite the dominance of the mouse in 2D user interfaces [Zhai, 1998], no device has been identified to be best suitable for all tasks in 3D user interfaces. For 2D desktop applications, many input devices have been designed and used; yet with proper mappings, they also work well in 3D [Bowman et al., 2004]. The mouse is generally faster than 3/6DOF devices for accurate 3D placement, despite the lack of a third DOF [Bérard et al., 2009]. In 2D user interfaces, the mouse is still one of the most widely used

devices and people are quite familiar with its form and function. Typically, the mouse is not designed to be used in immersive 3D environments such as HMDs, since it needs to be placed on a 2D surface to function properly. Naturally, direct manipulation with the hand is a very desirable method of interaction in virtual environments. Interestingly, in terms of input modalities for 3D manipulation, fingers are the most common devices on (multi-touch) mobile platforms. Consequently, some researchers have presented methods that use touch-based techniques for 3D manipulation of objects in virtual environments [Martinet et al., 2010].

Using an appropriate cursor representation significantly affects users' perception of the scene (and especially the cursor position) during selection. When working with stereoscopic content, the shape and behavior of a standard 2D cursor is not always suitable, as depth conflicts between the 3D positions of the cursor and the content can lead to confusion [Argelaguet et al., 2009]. A one-eyed (mono) cursor, first suggested in [Ware et al., 1997], eliminates stereo cue conflicts by displaying the cursor only to the dominant eye. However, the lack of a cursor visible in both eyes may cause some discomfort with long-term use [Hill et al., 2008]. Teather et al. found that one-eyed cursors improve screen-plane pointing/selection techniques [Teather et al., 2013]. Yet, for positioning techniques based on ray-casting, the one-eyed cursor performed significantly worse.

Manipulation techniques for virtual environments can be classified as exocentric and egocentric [Poupyrev et al., 1998]. In exocentric interaction, users interact with the 3D environment from the outside of it. For example, in the World in Miniature (WIM) technique, the user interacts with a small, handheld copy of the environment [Stoakley et al., 1995]. In egocentric manipulation, the user interacts from inside the environment. Ray-casting [Liang et al., 1993] is a common object selection technique which allows users to select an object by pointing at it with a virtual (pointer) ray. Ray-casting techniques work with both 2DOF devices, like the mouse, as well as 3/6DOF devices, such as 3DOF controllers. With a 3DOF input device, ray-casting is still (mostly) a 2DOF input technique, which is typically controlled (mostly) via two angular degrees of freedom (and 2DOF of movement) of the wrist during pointing. Some ray-casting variants use a 2D cursor and cast a ray through that cursor into the scene [Shaw et al., 1997] [Teather et al., 2013].

Sliding enables rapid positioning of objects in 3D scenes on a desktop system but is yet to be evaluated in an immersive system. It was also shown to work with 3D manipulation in VR such 3D puzzles [Machuca et al., 2018]. In this chapter, I focus on applying contact-plane sliding to VR and the comparison of different input devices and cursor displays. I choose to compare three input devices: mouse, the Vive controller, and the trackpad of the Vive controller. There are two conditions of cursor display: stereo cursor and one-eyed cursor. In the one-eyed cursor condition, users only see the cursor in their dominant eye.

I calibrated the virtual room to match the real one, so that the user could easily use the system sitting down in front of a desk wearing the HMD. I enabled full head tracking (movement and rotation). The implementation with the mouse, used on the desk surface, is similar to sliding on a desktop. While wearing the HMD, the user presses the left mouse button to select an object via image plane selection [Pierce et al., 1997]. The user then moves the mouse to change the cursor ray, which originated from a fixed camera position (the original headset position), therefore changing the position of the object. When the users release the mouse button, they “drop” the object at the last position.

In the trackpad condition, the user performs image plane selection by pressing down the trigger button on the controller. To account for the limited size of the trackpad, I map the movement of the cursor ray to relative movement on the trackpad, like on a normal laptop trackpad. The dragging motion of a finger is thus translated into a relative motion of the cursor (position control). Moving or rotating the controller does not change the cursor position. The cursor stays static when the user is not touching the trackpad. Again, the cursor ray originates from a fixed camera position, corresponding to the view at the start of the sliding movement. When the users release the trigger button, they “drop” the object at the last position.

In the controller-ray condition, which was named “hand-tracking” in the following published paper, the user can freely move and rotate the controller. The implementation was slightly different relative to the other two conditions, as the controller ray originates from the user’s hand and continues in the pointing direction of the controller. The user performs ray-casting selection by pressing the trigger button on the controller down. Moving or rotating the controller changes the controller ray, therefore changing the

object position. When the users release the trigger button, they “drop” the object at the last position.

In the mouse and trackpad condition, the cursor ray originates from the viewpoint at start of manipulation, and points to a cursor position on the screen. I extend the ray to intersect the scene and render a blue sphere at the intersection as the stereo cursor. Moving the mouse or swiping on the trackpad will then move the cursor on the screen, therefore changing the ray direction. Once an object selection is performed, the camera position is fixed for the duration of the positioning of that object. In the controller-ray condition, the ray originates from the controller in the user’s hand and points to the controller direction. In all three conditions, head movement or rotation does not change the ray, therefore does not change the object position. In the mouse condition, the users can rest their hand on the desk. In the trackpad condition, the controller pose does not affect object sliding. In the controller-ray condition, moving the controller forward and backward does not affect the controller ray. In all three conditions, users are able to perform object sliding in a comfortable pose.

I implemented both a stereo and a one-eyed cursor for object sliding in the Vive. For all input methods, the stereo cursor is rendered as a blue sphere along the cursor or controller ray that always snaps to the scene geometry. For the one-eyed cursor condition, I only display the cursor to the dominant eye of the user. This condition does not provide cursor depth cues and the cursor stays on a fixed plane orthogonal to the original camera direction in this condition.

This chapter addresses the third and fourth research questions: “**How do input devices compare for object sliding in VR?**” and “**Which cursor display performs better for object sliding in VR?**” I performed a user study that compared the performance of input devices and cursor displays in a 3D object sliding task. I hypothesized that the mouse would outperform the controller-ray and the trackpad, due to the 2DOF nature of sliding and users’ familiarity of the mouse. I also hypothesized that stereo cursor would be better in terms of performance, due to the potential eye fatigue introduced by one-eyed cursor.

I had ethics approval for the user study I performed, and participants signed consent forms. All the participants were informed about the potential risk of motion

sickness. The study results confirmed the first hypothesis. Overall, the mouse performed in general better than both the controller-ray and trackpad conditions. Interestingly, in both controller-ray and trackpad conditions, some users tried to use the second hand to stabilize their dominant hand, even when sitting in VR. This is the participants' way of trying to avoid the Heisenberg effect [Bowman et al., 2001], where the slight movement of pressing or releasing a button affects the accuracy of the manipulation. The mouse did not suffer from the same problem. The 2DOF nature of the contact-plane sliding technique matches the affordances of 2DOF input devices such as the mouse. This could give an advantage to the mouse, where the 2DOF mouse movement is constrained by the desk surface and to use the desk as a stable reference system. Yet, the result confirms that the mouse is a good input device for precise 3D positioning in an HMD-based VR system in situations where users have a stable surface for the mouse available, such as a desk or a chair-integrated mouse pad. This finding could be a helpful guideline to future 3D modelling applications in VR.

Stereo cursor and one-eyed cursor did not exhibit a significant difference in performance. Yet, users were more comfortable using the stereo cursor for 3D positioning in the HMD. When designing an interface for HMD, both my results as well as the results of previous work [Teather et al., 2013] have to be taken into consideration. I speculate that a stereo cursor would outperform a one-eyed cursor in a combined task requiring both selecting and positioning, but this needs to be verified by a future study. The details about the implementation of the contact-plane sliding technique in VR, experimental design, user studies, and study results can be found in the following published paper.

The work was previously published in a full paper titled "Comparing Input Methods and Cursors for 3D Positioning with Head-Mounted Displays" at the ACM Symposium on Applied Perception conference in 2018. I co-authored this paper with Wolfgang Stuerzlinger and Bernhard Riecke.

Abstract

Moving objects is an important task in 3D user interfaces. In this work, we focus on (precise) 3D object positioning in immersive virtual reality systems, especially head-

mounted displays (HMDs). To evaluate input method performance for 3D positioning, we focus on an existing sliding algorithm, in which objects slide on any contact surface. Sliding enables rapid positioning of objects in 3D scenes on a desktop system but is yet to be evaluated in an immersive system. We performed a user study that compared the efficiency and accuracy of different input methods (mouse, hand-tracking, and trackpad) and cursor display conditions (stereo cursor and one-eyed cursor) for 3D positioning tasks with the HTC Vive. The results showed that the mouse outperformed hand-tracking and the trackpad, in terms of efficiency and accuracy. Stereo cursor and one-eyed cursor did not demonstrate a significant difference in performance, yet the stereo cursor condition was rated more favorable. For situations where the user is seated in immersive VR, the mouse is thus still the best input device for precise 3D positioning.

CCS Concepts

Human-centered computing → Human computer interaction (HCI).

Keywords

cursors, input devices, 3D positioning.

4.1. Introduction

Posing a 3D rigid object, i.e., manipulating the position and orientation of an object, is a basic task in 3D user interfaces. This task can be time-consuming as 6 degrees of freedom (DOFs) must be controlled: 3 DOFs for translation along three axes and 3 DOFs for rotation around three axes.

Input devices are an important component in designing, developing, and using 3D user interfaces. There are many different types of input devices to choose from and some of them may be more appropriate for certain tasks than others. Despite the dominance of the mouse in 2D user interfaces [Zhai, 1998], no device has been identified to be best suitable for all tasks in 3D user interfaces. For 2D desktop applications, many input devices have been designed and used; yet with proper

mappings, they also work well in 3D [Bowman et al., 2004]. The mouse is still one of the most widely used devices in 2D user interfaces and people are quite familiar with the form and function of the mouse.

Typically, the mouse is not designed to be used in immersive 3D environments, since it needs to be placed on a 2D surface to function properly. Naturally, direct manipulation with the hand is a very desirable method of interaction for virtual environments. Interestingly, on (multi-touch) mobile platforms fingers are the most common input devices for 3D manipulation. Consequently, some researchers have presented methods that use touch-based techniques for 3D manipulation in virtual environments [Martinet et al., 2010].

Using an appropriate cursor representation significantly affects users' perception of the scene (and especially the cursor position) during selection. When working with stereoscopic content, the shape and behavior of a standard 2D cursor is not always suitable, as depth conflicts between the 3D positions of the cursor and the content can lead to confusion [Argelaguet et al., 2009]. A one-eyed (mono) cursor, first suggested by Ware and Lowther [Ware et al., 1997], eliminates stereo cue conflicts by displaying the cursor only to the dominant eye. However, the lack of a cursor visible in both eyes may cause some discomfort with long-term use [Hill et al., 2008]. Teather et al. [Teather et al., 2013] found that one-eyed cursors improve screen-plane pointing/selection techniques. Yet, for positioning techniques based on ray-casting, the one-eyed cursor performed significantly worse.

Virtual reality (VR) headsets have become rapidly prevalent in recent years and are starting to be widely used with computer games. More game developers than ever are helping with the VR revolution in the game industry. The HTC Vive was officially unveiled in 2016 and has become one of the best VR headsets in the industry. The Vive includes two wireless handheld controllers. Figure 4.1 shows a person using the trackpad on the Vive controller. Hundreds of games and applications are now available for the HTC Vive. For both games and VR applications, 3D positioning is a fundamental task.

Although we know that the mouse is an ideal input device for 2D interfaces and some 3D interfaces [Zhai, 1998], there has been little research that evaluated the

suitability of the mouse for 3D positioning tasks in head-mounted displays (HMDs). Moreover, although the one-eyed cursor was shown to be beneficial for 3D selection [Teather et al., 2013], it remains to be seen whether it performs well for 3D positioning.

We are interested in the efficiency and accuracy of input methods and cursors for 3D positioning with HMDs. We choose a sliding algorithm, as it works well with a mouse in desktop systems [Oh et al., 2005]. As this kind of positioning algorithm requires only 2D input, it should also work well with ray-casting. With a seated user wearing an HMD, we hypothesize that the mouse would thus be faster and more accurate than 3D input devices (hand-tracking with the Vive controller) and touchpads (trackpad on the Vive controller) for precise 3D positioning. Due to the potential eye fatigue and discomfort introduced by one-eyed cursor, we also hypothesize that the stereo cursor would perform better than a one-eyed cursor for 3D positioning with HMDs.

Figure 4.1. The HTC Vive controller. The person's thumb is touching the trackpad.

We measure completion time and error in a 3D positioning task and collect data from a usability questionnaire. Besides the main effect of input method and cursor display, we also want to analyze if the type of scene surfaces (smooth or irregular) and object density (empty or cluttered) in the scene have a significant impact on user performance. We hypothesize that the type of surfaces would not influence user performance, as sliding is robust to irregular surfaces [Oh et al., 2005]. We also hypothesize that cluttered scenes would yield longer completion time than empty ones,

simply because users might (need to) slide around obstacles to find a path to the target position in a cluttered scene. Analyzing the influence of scene surfaces and object density could yield some interesting insights about the differences between input techniques and should not affect the main effects of input method or cursor display.

In this paper, we first review relevant object manipulation methods with various input devices. Then we discuss the sliding algorithm. Subsequently, we describe our user study, including the implementation of sliding in the HTC Vive, with both stereo and one-eyed cursors. Finally, we discuss the results and mention potential future work.

4.2. Related Work

There has been substantial research in the field of 3D manipulation. Butterworth et al. introduced a 3D modelling program to be used in an HMD [Butterworth et al., 1992]. They used a 6D handheld mouse as the input device. Besançon et al. compared the mouse, tactile, and tangible input for 3D manipulation [Besançon et al., 2017]. They found that the three input modalities provide the same level of accuracy, yet tangible input is the fastest. Krichenbauer et al. compared virtual reality and augmented reality for 3D manipulation [Krichenbauer et al., 2018]. They found that 3D manipulation was more efficient in virtual reality than augmented reality. They also compared the mouse and a 3D input device for 3D manipulation and found no significant difference. Hoppe et al. did a survey on various input and output devices and associated interaction techniques for 3D interaction [Hoppe et al., 2017]. Some 3/6DOF devices perform better than the mouse on specific tasks, e.g., the Control Action Table [Hachet et al., 2003], the GlobeFish and GlobeMouse [Froehlich et al., 2006]. Yet, the mouse is generally more efficient than 3/6DOF devices for accurate 3D placement, despite the lack of a third DOF [Bérard et al., 2009].

Mouse-based 3D manipulation is not without its limitations. First, simultaneous translation along all three directions is not possible, due to the 2D nature of the device. This can be compensated through proper mapping and use of constraints, e.g., [Sun et al., 2016]. Second, 3D rotations are not supported efficiently. Some techniques limit the rotations to a single axis at a given time. Others enable simultaneous manipulation of two axes, e.g., [Zhao et al., 2011].

Manipulation techniques for virtual environments can be classified as exocentric and egocentric [Poupyrev et al., 1998]. In exocentric interaction, users interact with the 3D environment from the outside of it. For example, in the World in Miniature (WIM) technique, the user interacts with a small, handheld copy of the environment [Stoakley et al., 1995]. In egocentric manipulation, the user interacts from inside the environment. Virtual hand and virtual pointer are the two main metaphors for egocentric manipulation. The Go-Go technique extends the virtual hand's reaching distance through a non-linear mapping function applied to the user's real hand extension [Poupyrev et al., 1996]. It allows direct seamless 6DOF object manipulation. Ray-casting allows users to select an object by pointing at it with a virtual (pointer) ray. Ray-casting is a good technique for object selection, but not necessarily for object manipulation. Thus, the HOMER technique used ray-casting for object selection together with hand-centered manipulation [Bowman et al., 1997].

Various cursor display methods have been proposed for 3D selection. The silk cursor technique used a semi-transparent volume cursor, which provided additional depth cues through occlusion [Zhai et al., 1994]. In the evaluation of the silk cursor the authors compared mono and stereo display. They found that stereo display performed significantly faster than mono display in a selection task. Vanacken et al. introduced the 3D bubble cursor, a semi-transparent sphere that dynamically resizes to only enclose the closest target [Vanacken et al., 2007]. The 3D bubble cursor was effective for both sparse and dense environments and it outperformed the 3D point cursor in a selection task. Jáuregui et al. proposed two new 3D cursor metaphors controlled by 2D input devices: The Hand Avatar and The Torch [Jáuregui et al., 2012]. These two metaphors explored both image-based and projection-based cursor visualization techniques. The user evaluation showed that both 3D cursors significantly increased users' depth perception, but at the expense of an increase of the selection time and a decrease of accuracy.

Previous research has demonstrated that the mouse is a reliable input device for 3D manipulation in desktop systems and is well suited for the most fundamental and frequent task (object selection and placement) [Bérard et al., 2009]. However, it is important to evaluate the mouse for placement tasks in immersive systems, as little previous research has investigated this option. Various 2D/3D cursors have been

proposed and evaluated for 3D selection, yet the best cursor for 3D positioning has not been identified.

Oh et al. [Oh et al., 2005] presented a sliding algorithm for desktop systems, with a mouse as input device, where the object follows the cursor position directly and slides on any surface behind it, i.e., the moving object always stays attached to other objects. Their user study showed that sliding was easier to understand and significantly improved the efficiency of object manipulation in CAD systems. SHIFT-Sliding and DEPTH-POP are two recently introduced techniques that (i) generalize sliding to support floating and interpenetrating objects, (ii) address the inherent depth ambiguity, and (iii) significantly speed up common 3D positioning tasks [Sun et al., 2016]. The authors identified that the techniques could potentially be used with other input devices and platforms but did not evaluate sliding in an immersive environment. Sliding has an intuitive mapping of input to object movement; thus, we hypothesize that it would also perform well with HMDs and would be a good choice for evaluating input methods and cursor display methods.

4.3. Sliding

Sliding maps object movement so that the manipulated object moves along the surface behind it that it is currently in contact with [Oh et al., 2005]. We use the normal vector at the contact point to determine the sliding plane. With this contact constraint, we can directly map 2D motions of the mouse cursor to 3D movement of the object. Figure 4.2 illustrates sliding. When the user selects an object (at position A), we record the intersection of the mouse ray on the object as the start point. The starting point and the normal vector define the sliding plane. The intersection of a new mouse ray and the sliding plane becomes the end-point of the object translation. By moving the mouse cursor, the user then effectively translates the object parallel to the sliding plane.

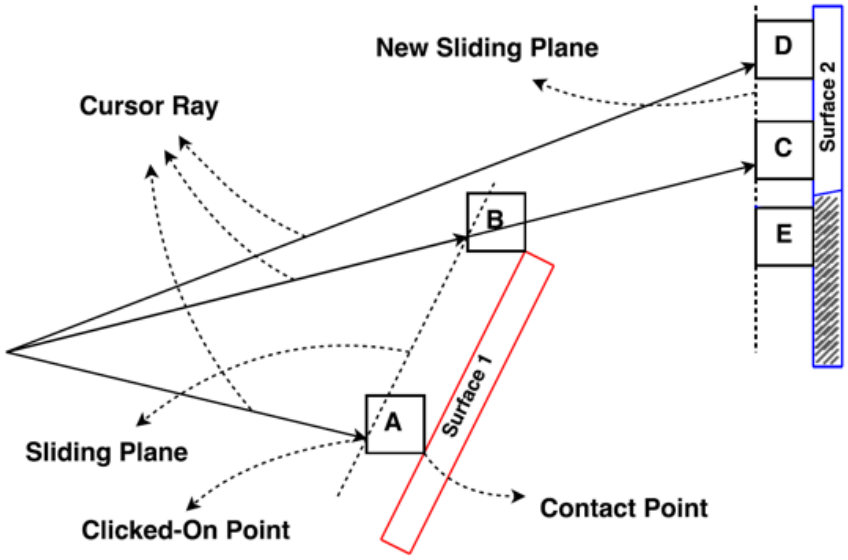


Figure 4.2. Illustration of sliding movements for an object across the front surfaces of two objects with an upwards mouse movement (positions A-D). The shaded part of surface 2 is occluded by surface 1. Position E can only be reached from C with a downwards mouse movement. (Figure from Sun et al. [Sun et al., 2016]).

4.4. User Study

We performed a user study that compared three input methods and two cursor displays for object sliding in HTC Vive.

4.4.1. Participants

We recruited 12 (8 female) undergrad and graduate students from the local university population. Undergraduate students received course credit for participation. We did not screen participants for 3D/VR experience. Our participants had varying gaming expertise, with 25% playing games regularly. All the participants were informed about the potential risk of motion sickness.

4.4.2. Apparatus

The hardware setup for the experiment used an HTC Vive (headset, base stations, controllers), mouse and keyboard, a 27-inch monitor, and a desktop computer. The Vive HMD has a diagonal field of view of 110 degrees, a display resolution of 1080x1200, and a refresh rate of 90 Hz. We used a desktop computer with 3.6 GHz i7 processor, 16 GB of memory, a NVIDIA GeForce GTX 1080 graphics card, and Unity 5.5 for the implementation. We used the monitor to observe the users' actions during the experiments.

There was a 2-minute training session before each condition, which introduced participants to the techniques in a playground environment, which did not include any version of the experimental tasks.

4.4.3. Implementation

We adapted the sliding algorithm to enable its use with a VR HMD. We used three different input methods: the mouse, hand-tracking with the Vive controller, and the trackpad on the Vive controller. For each input method, the user had to wear the Vive headset. We calibrated the virtual room to match the real one, so that the user could easily use the system sitting down. Full head tracking (movement and rotation) was enabled. The scale of the virtual environment was significantly bigger than the typical head movements of a user in a seated posture, therefore head movements can essentially be ignored in our experiment.

The implementation with the mouse was similar to sliding on a desktop. While wearing the HMD, the user sat in front of the desk and used the mouse (on the desk surface) to change the cursor ray, which originated from the camera position. The user pressed the left button for selection, but the space bar for confirmation of object placement.

To account for the limited size of the trackpad, we mapped the movement of the cursor ray to relative movement on the trackpad, like on a normal laptop touchpad. The dragging motion of a finger is thus translated into a relative motion of the cursor. The cursor stays static when the user is not touching the trackpad. We fine-tuned the control-

display rate in a pilot study. Again, the cursor ray originated from the camera/head position. The translation of the selected object depended on the intersection of the new cursor ray with the sliding plane.

For hand-tracking with the controller, the user could move and rotate the controller freely. The implementation was slightly different, as the controller ray originates from the user's hand and continues in the pointing direction of the controller. For stereo and one-eyed cursor, we used a different cursor ray for sliding. We discuss this in the next two paragraphs. For both hand-tracking and trackpad, the user used the trigger button in the bottom of the Vive controller to select an object.

We implemented both stereo cursor and one-eyed cursor for sliding in the Vive. For all input methods, the stereo cursor was rendered as a blue sphere along the controller or cursor ray that always snaps to the scene geometry. In the stereo cursor condition, the user can see the cursor in both eyes. For the mouse and the trackpad, the cursor ray originated from the camera position. With hand-tracking, the controller ray originated from the user's hand, and we used the intersection of the controller ray and the sliding plane to determine the next position of the object.

For the one-eyed cursor condition, we only displayed the cursor to the dominant eye of the user. This condition does not provide depth cues. We again used a blue sphere as the cursor, yet in this condition the cursor stays on a plane orthogonal to the original camera direction. We placed this plane close to the camera, as we do not want the sphere to be occluded by objects in the scene. For the mouse and the trackpad, the sphere position was derived from the intersection of the cursor ray with the plane. For hand-tracking, the sphere was placed at the intersection of the controller ray with the plane. For sliding, we extended the cursor ray from the camera position to the sphere and used that during interaction.

4.4.4. Experiment Design

We designed a 3D object positioning experiment and asked participants to move an object to a target position in various scenes. When the user positioned the object at the target position, they pressed the space bar on the keyboard (in mouse condition) or the menu button on the Vive controller (in hand-tracking and trackpad conditions) for

confirmation. In the mouse condition, participants could use their other hand to press the space bar. We measured the completion time and error distance from the ideal target position. Completion time was measured in seconds. The error measure was calculated as the absolute distance to the target over the object size. We recorded all actions of each user. The experiment had a 3 (input method) x 2 (cursor display) x 2 (surface) x 2 (object density) within-subject design. We counterbalanced the order of input method, cursor display, surface, and object density conditions to avoid learning effects. Figures 3 to 5 illustrate the three input methods.

Figure 4.3. A user performing the task with the mouse. Note the other hand on the space bar for confirming placement.

Besides the main effect of input method and cursor display, we also wanted to analyze if the type of surface and object density in the scene influence user performance. Therefore, we used a 2 (surface) x 2 (object density) design for the tasks. Figure 4.6 shows one sample scene for each combination of surface and object density. The two scenes we used were a room and a terrain, which represented smooth and irregular sliding surfaces. There are multiple paths to move the object to the target position. In the scenes with irregular surfaces, the object often collides with the rest of the scene. The sliding algorithm then “pops” the object automatically towards the viewer to resolve the collision, which obviates the need for manual collision resolution. The two conditions for object density were empty and cluttered. In the cluttered scenes, the target position is usually partially hidden. In this case, the users had to slide around obstacles to position the object correctly.

Figure 4.4. A participant performing the task with hand-tracking with the controller.

Figure 4.5. A participant performing the task with the trackpad on the controller.

Each task condition had 5 trials. The target positions were all in contact with the scene. We displayed the scenes in front of the users, so the users do not have to perform head rotations. This reduces any potential confound that could be introduced by such head rotations, e.g., if the user loses sight of the target position. Each user performed all trials, corresponding to a total of 120 (5x3x2x2x2) trials for each user. We asked the participants to perform the tasks as quickly and as accurately as possible.

After the users finished all the tasks, we asked them to answer questions about the usability of the different input and cursor display methods. The users had to rate the ease of use, ease of learning, and level of comfort for each input and cursor display method. The ratings used a seven-level Likert scale. We also asked participants for their

favorite and least favorite input and cursor display method. Finally, we asked them what could be done to improve the interaction further.

The total duration of the experiment varied from 45 minutes to one hour for each participant.

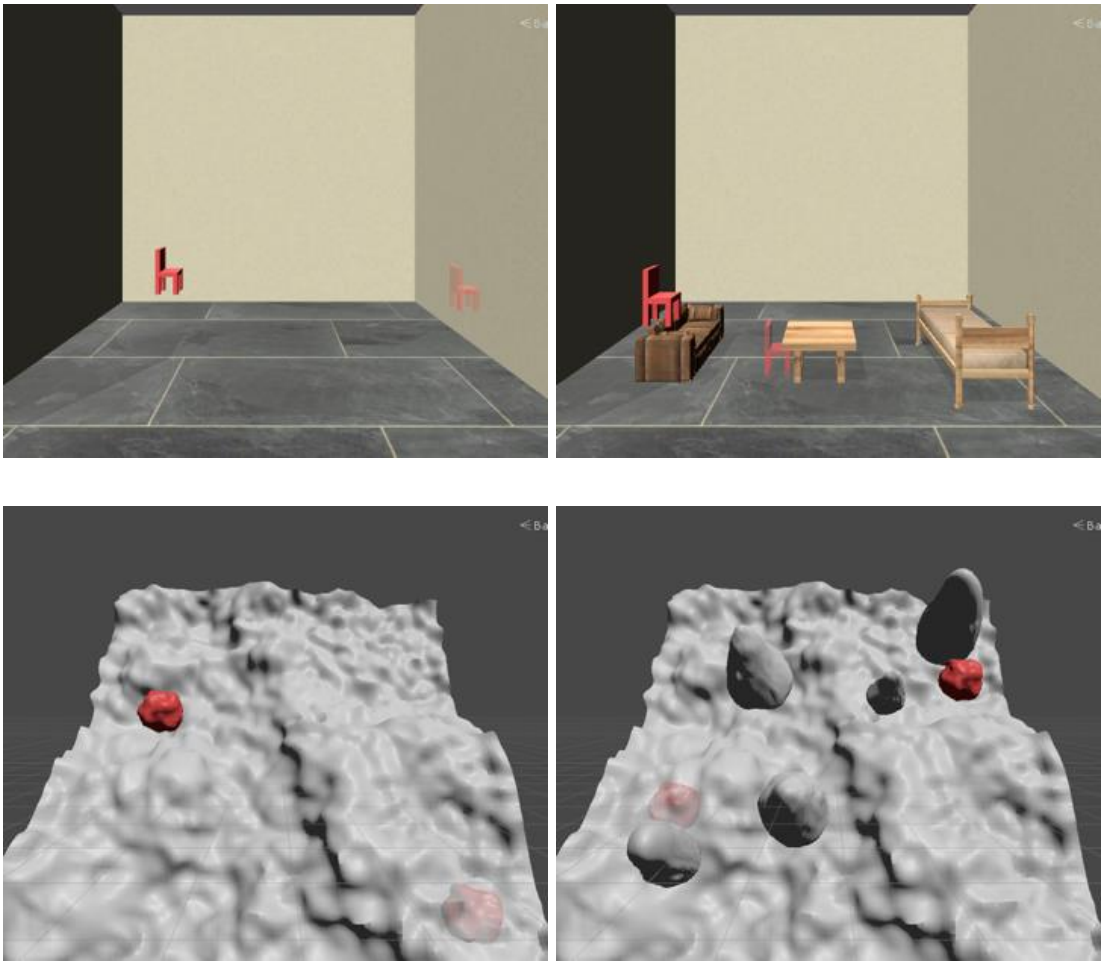


Figure 4.6. Top: Smooth-Empty, Smooth-Cluttered conditions. Bottom: Irregular-Empty, Irregular-Cluttered conditions. The target positions are rendered as semi-transparent.

4.4.5. Results

Shapiro Wilk tests were conducted to assess the normality of the dataset. The results showed that the data of neither completion time nor error measure was normally distributed for any combination of input method and cursor display ($p < .001$). Therefore,

we ran an aligned rank transform (ART) [Wobbrock et al., 2011] on the data followed by a within-subject ANOVA on the ranks and report the statistical results accordingly.

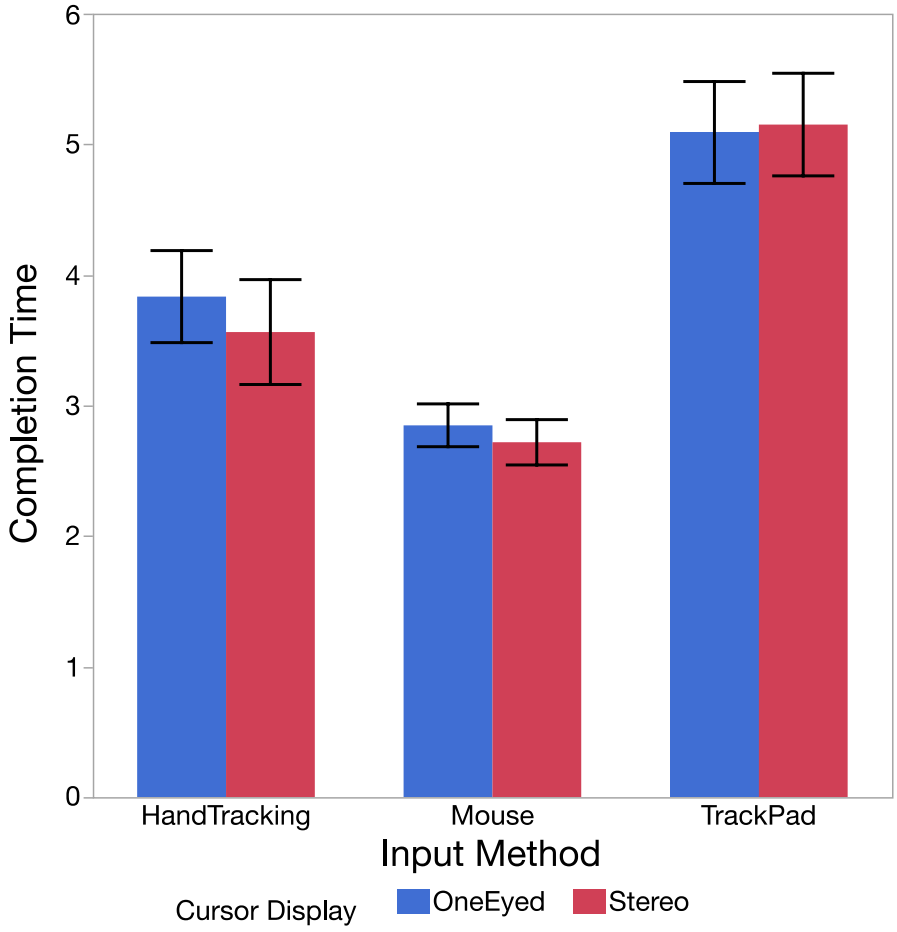


Figure 4.7. Average completion times (in seconds) for input and cursor display method. Each error bar is constructed using a 95% confidence interval of the mean.

The ANOVA results showed that input method had a significant effect on completion time, $F(2, 22) = 50.29$, $p < .0001$. The Tukey's post hoc analysis showed that the mouse ($M = 2.78$, $SD = 1.26$) was significantly faster than hand-tracking ($M = 3.70$, $SD = 2.82$), and hand-tracking was significantly faster than the trackpad ($M = 5.12$, $SD = 2.92$). The 95% confidence intervals of the three conditions also do not overlap. Stereo cursor ($M = 3.81$, $SD = 2.72$) and one-eyed cursor ($M = 3.92$, $SD = 2.54$) did not show a significant difference on completion time $F(1, 11) = 1.12$, $p > .05$. The interaction of input method and cursor display was not significant, $F(2, 22) = 0.30$, $p > .05$. See Figure 4.7.

Surface type had a significant effect on completion time, $F(1, 11) = 19.04$, $p < .005$, where the irregular surface ($M = 3.34$, $SD = 1.76$) was faster than the smooth surface ($M = 4.40$, $SD = 3.19$). Object density had a significant effect on completion time ($F(1, 11.01) = 15.49$, $p < .005$), where the empty scene ($M = 3.68$, $SD = 2.31$) was faster than the cluttered scene ($M = 4.10$, $SD = 2.97$).

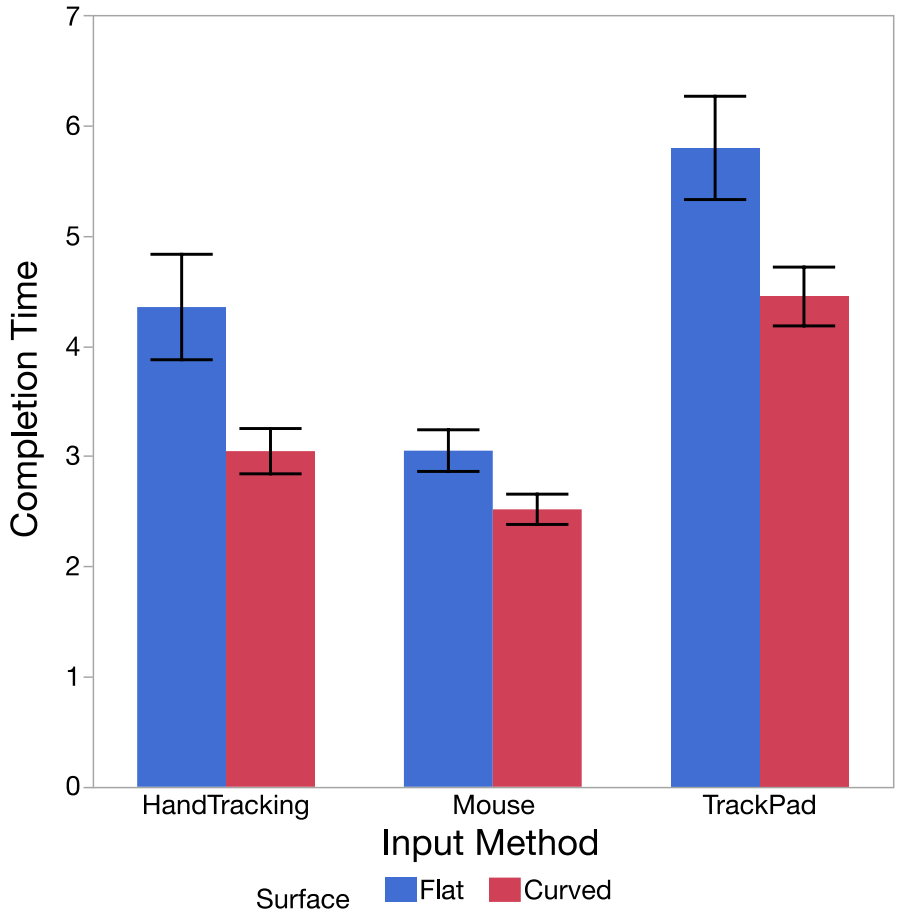


Figure 4.8. Average completion times (in seconds) for input method and surface type. Each error bar is constructed using a 95% confidence interval of the mean.

The interaction of input method and surface was significant on completion time, $F(2, 22) = 3.95$, $p < .05$, with the combinations of Mouse-Smooth, Mouse-Irregular, and HandTracking-Irregular, being significantly faster than the other three combinations, see Figure 4.8. The interaction of surface and object density was also significant on

completion time ($F(1, 11.03) = 17.88$, $p < .005$), with Smooth-Cluttered being the slowest, see Figure 4.9. All the other interactions were not significant.

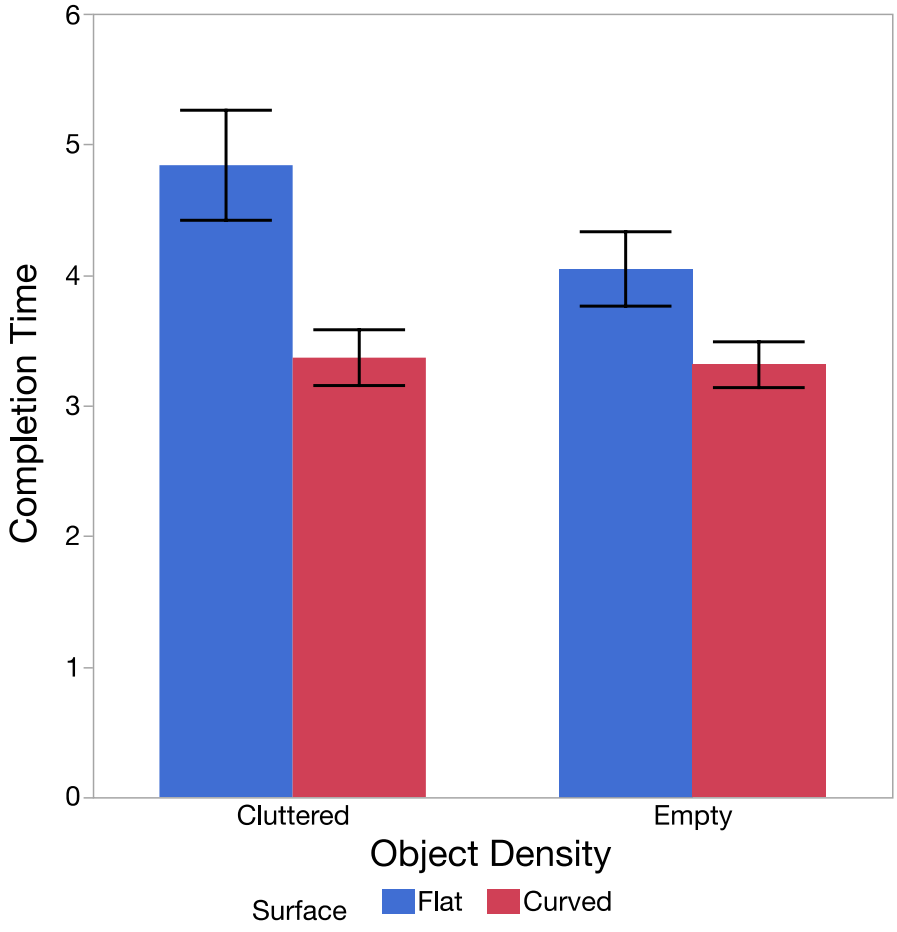


Figure 4.9. Average completion times (in seconds) for object density and surface type. Each error bar is constructed using a 95% confidence interval of the mean.

In terms of the error measure, input method had a significant effect, $F(2, 22) = 8.33$, $p < .005$. The Tukey's post hoc analysis showed that the mouse ($M = 0.103$, $SD = 0.128$) was significantly more accurate than trackpad ($M = 0.152$, $SD = 0.179$). Hand-tracking ($M = 0.143$, $SD = 0.151$) was not significantly different from mouse or trackpad. Stereo cursor ($M = 0.128$, $SD = 0.147$) and one-eyed cursor ($M = 0.138$, $SD = 0.164$) did not yield a significant difference on the error measure, $F(1, 11) = 0.21$, $p > .05$. The interaction of input method and cursor display was not significant, $F(2, 22.3) = 2.42$, $p > .05$. See Figure 4.10.

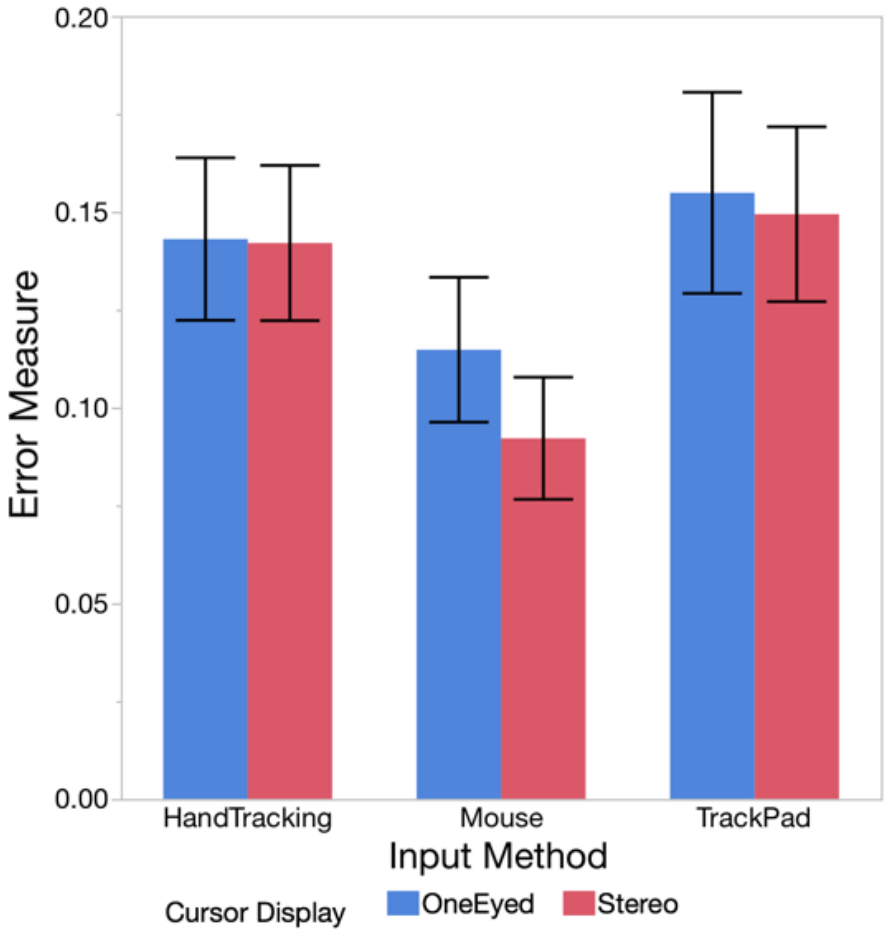


Figure 4.10. Average error measures for input and cursor display method. Error measure was calculated as the absolute distance to the target over the object size. Each error bar is constructed using a 95% confidence interval of the mean.

Surface type had a significant effect on the error measure, $F(1, 11) = 52.49$, $p < .0001$, where the irregular surface ($M = 0.090$, $SD = 0.095$) was more accurate than the smooth surface ($M = 0.175$, $SD = 0.189$). The empty scene ($M = 0.127$, $SD = 0.142$) was significantly more accurate than the cluttered scene ($M = 0.144$, $SD = 0.171$), $F(1, 11.01) = 12.62$, $p < .01$.

The interaction of input method and object density was not significant for error measure, $F(2, 22.19) = 3.09$, $p > .05$. See Figure 4.11. The interaction of surface type and object density was not significant for the error measure, $F(1, 11.07) = 4.18$, $p > .05$. All the other interactions were also not significant on error measure.

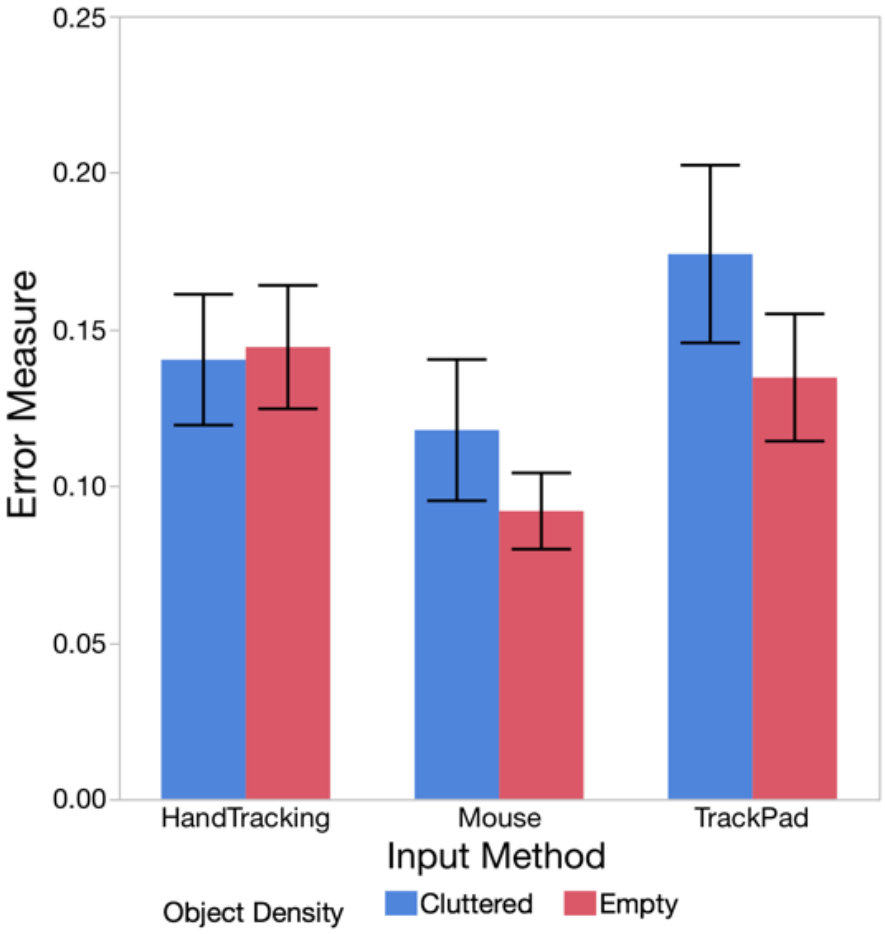


Figure 4.11. Average error measures for input method and object density. Error measure was calculated as the absolute distance to the target over the object size. Each error bar is constructed using a 95% confidence interval of the mean.

Eleven out of 12 participants found the mouse easy to use, 9 out of 12 found hand-tracking easy to use, and 7 found the trackpad easy to use. All twelve participants found the mouse easy to learn, 10 found hand-tracking easy to learn, and 9 found the trackpad easy to learn. All participants found the mouse comfortable to use, 10 found hand-tracking comfortable to use, and only 7 found the trackpad comfortable to use.

Six participants rated the mouse as their favorite input method among the three, and 4 participants liked hand-tracking the most. 7 participants rated trackpad as their least favorite, while 5 rated hand-tracking as their least favorite.

All participants found the stereo cursor easy to use, and 10 found the one-eyed cursor easy to use. All participants found the stereo cursor easy to learn, and 11 found the one-eyed cursor easy to learn. All participants found the stereo cursor comfortable to use, and 10 found the one-eyed cursor comfortable to use.

Eleven out of 12 participants rated the stereo cursor as their favorite cursor display method, and 1 participant did not have a preference.

4.5. Discussion

Results showed that the mouse performed the best in terms of completion time for sliding tasks in the HTC Vive, which supports our first hypothesis in terms of time. In terms of error measure, the mouse was significantly more accurate than the touchpad (but not the hand-tracking condition), which partially supports our first hypothesis. The participants commented that they liked the mouse as it is fast and precise, and they were more familiar with it as they used it daily. Our results might also be partially explainable by the fact that when using the mouse, one can rest the hand using the desk surface, which serves as a stable spatial reference. Such stabilization is not as easy with a single hand in the hand-tracking condition. Thus, it is not surprising that approximately half of the participants used the other hand to stabilize the controller pose while pointing. Interestingly, in the trackpad condition, the controller itself serves as a spatial reference for the thumb interaction. Another reason that could partially explain the mouse being fastest is that it requires less physical movement than hand movements with the controller. However, the trackpad typically requires even less physical movement, but still ended up being the slowest device.

Some participants stated that hand-tracking was easy and natural to control. However, hand jitter made it sometimes hard to position the object precisely. We observed that hand jitter can cause problems in some of our tasks, where the object had to be placed in a specific position, which required higher precision in terms of hand orientation. In this kind of situation, a subset of participants chose to stabilize the controller with both hands to be more accurate. Hand-tracking also produced higher fatigue, likely because it required on average larger hand movements.

Participants commented that it was difficult to be precise with the trackpad. They were not comfortable with using the thumb on such a small input space. Some suggested that using one-to-one position control for the trackpad might help. Yet, this would be difficult to do due the limited trackpad size of the device. Conversely, with the mouse one can use both the wrist and the fingers to accurately position the device.

Our results suggest that the mouse is a reliable input device for (precise) 3D positioning in HMDs, which matches the conclusion of Bérard et al. [Bérard et al., 2009]. Thus, there is a need for designing better user interfaces for situations, where the user can use a mouse, such as being seated or in front of a standing desk. One good alternative for a seated user is a rotatable ergonomic chair with integrated mouse pad, such as the Mobo chairs.

We choose to implement sliding as the main interaction method. Sliding is essentially a 2DOF technique, as object movement is constrained by the surface it is in contact with [Oh et al., 2005]. This could give an advantage to 2DOF input devices such as the mouse, where the 2DOF mouse movement is constrained by the desk surface and to use the desk as a stable reference system. Naturally, 2DOF devices provide intuitive input mappings for sliding, but note that (3D) ray-casting is also (mostly) a 2DOF input technique, corresponding to the two angular degrees of freedom used during pointing. To generalize the results, evaluation with a full 3DOF positioning technique is necessary.

Cursor display did not have a significant effect on either completion time or the error measure, which rejects our second hypothesis. However, eleven out of twelve participants preferred the stereo cursor over the one-eyed cursor, as the one-eyed cursor produced more eye fatigue. Teather et al. [Teather et al., 2013] showed that the one-eyed cursor benefits 3D selection. However, unlike a Fitts' Law experiment [MacKenzie, 1992], we did not measure object selection time in the tasks. The timing started the instant when the users selected the object. Overall, we found that in a positioning task, different cursor display methods did not make a difference on performance. The one-eyed cursor may not be as comfortable to use, yet its lack of support for accurate depth perception does not seem to have a negative impact on the positioning time. We guess that this is likely due to the fact that users focus on the

moving object during positioning tasks and not the cursor itself, as observed in previous work [Oh et al., 2005].

A (relatively) empty scene required less time to complete tasks than a cluttered one, which matches our hypothesis. This is not surprising, as users naturally slide objects around obstacles. The participants also mentioned that they found this to be the easiest method to reach the target position with an object.

Interestingly, tasks took longer in the smooth scenes (room) than in the irregular scenes (terrain). One potential reason is that in the room scenes, the partially hidden targets were in closer proximity to the rest of the scene, which caused more collisions. In such situations, users had to reposition the object when it was popped to the front to avoid such collisions. Still. this result shows that sliding is robust to the surfaces of the scene.

4.6. Conclusion

We compared three input and two cursor display methods for precise positioning in the HTC Vive. The mouse performed in general better than both hand-tracking and the trackpad. As the 2DOF nature of the technique matches 2DOF input devices such as the mouse, this result might partially due to the use of a sliding technique we choose for our evaluation. Yet, we believe that the result confirms that the mouse is a good input device for precise 3D positioning in an HMD-based VR system in situations where users have a stable surface for the mouse available, such as a table or a chair-integrated mouse pad.

Cursor display did not have a significant effect on either completion time or error measure. Yet, users were more comfortable using the stereo cursor for 3D positioning in the HMD. When designing an interface for HMD, both our and Teather et al.'s [Teather et al., 2013] results have to be taken into consideration. We presume that stereo cursor would outperform one-eyed cursor in a combined task requiring both selecting and positioning, but this needs to be verified by a future study.

Some may argue that the mouse is not suitable for immersive virtual reality. However, we believe that it is currently not practical to expect people to use an HMD in a

standing pose for an extended period of time, say for a full workday. Therefore, there is a need for designing better user interfaces for users that use an HMD in a seated posture, or at least in front of a standing desk. The choice of input device depends on the tasks and platforms. With a sliding technique, the mouse is the better device. To generalize the results, we plan to evaluate positioning in a comparative study involving a full 3DOF technique.

In the future, we also plan to look at 3D positioning tasks that require bigger head or body movements. Moreover, we plan to implement SHIFT-Sliding and DEPTH-POP for the HTC Vive to enable full 3D positioning in more general scenes [Sun et al., 2016]. To improve the precision of positioning, non-linear mappings for the trackpad are another avenue to explore. Finally, we are considering another study that records the time for the movement and fine-tuning phases separately, to better analyze how the movement is affected by the input device.

Chapter 5.

Extended Sliding in VR

In this chapter, my goal is to further improve 3D positioning performance in VR systems. Although precise 3D positioning is not always necessary in virtual environments, e.g., in VR games, it is still an important task for applications such as 3D modelling in VR. In this part of my work, I focus on 3D positioning techniques in immersive environments that use a 3DOF controller as input device. My motivation is to improve the speed and accuracy of 3D positioning techniques in such an environment. Towards this goal, I adapted the extended sliding technique to VR systems with a VR controller as input device and compared it with previously presented 3DOF positioning techniques. The results showed that extended sliding significantly improved the accuracy for 3D positioning tasks for targets that were in contact with the scene.

5.1. Introduction

Although the mouse performed well in the 3D positioning task discussed in Chapter 4, it is not the most natural input device for a VR system that uses an HMD, which permits the user to move around. The mouse is only usable as an input device when there is a stable surface, e.g., when users are seated at a desk or in front of a standing desk. For situations where the user is wearing an HMD and is standing or sitting on a swivel chair within a free area, the user needs to reply on mid-air manipulation, where 3DOF/6DOF input devices provide more natural input mappings.

For objects that are beyond arm's reach, finding an appropriate mapping from 3D hand movements to 3D object movements is challenging. There have been various proposals for 3D mid-air manipulation mappings. Some metaphors apply only to objects that are within reach of the users' hand, e.g., the virtual hand technique [Bowman et al., 2004]. In this kind of technique, users intersect the virtual representation of their hand/controller with the object to select and directly manipulate the object. The

translation and rotation of the hand is then directly mapped to the translation and rotation of the object, which makes this technique well suited for coarse manipulation. For manipulating distant objects, the Go-Go technique [Poupyrev et al., 1996] extends the reach of the virtual hand non-linearly. The HOMER technique uses ray-casting for object grabbing followed by hand-centered manipulation [Bowman et al., 1997]. The authors of this work also introduced ray-casting with reeling to add a single degree of freedom, making it possible to change the length of the ray, which allows the user to change at which distance the object is placed. They performed a user study that compared Go-Go and ray-casting techniques, and the results demonstrate that both techniques have their strengths and weaknesses [Poupyrev et al., 1998]. They identified that ray-casting (without reeling) performs better only in positioning tasks that do not require a (substantial) change of object distance relative to the viewer.

Identifying a mid-air VR manipulation technique that achieves satisfactory precision is still an open problem [Mendes et al., 2019]. The relative instability of the hand contributes to a loss of accuracy. Some techniques attempted to scale down the control/display ratio from hand movement to object movement to increase precision. The PRISM technique [Frees et al., 2005] [Frees et al., 2007] uses the hand speed of the user to gradually switch between different modes, with different control/display ratios. When the ratio was reduced, this introduces an offset between hand and object positions. Although the authors presented some methods to remove/reduce this offset, the need to change mode made them focus more on precision in the precision/speed trade-off. Similarly, the scaled HOMER technique scaled down the user's hand movement to improve precision [Wilkes et al., 2008]. It also introduced an offset between hand movement and object position, and thus the object does not always stay "in" the user's hand. For distant object manipulation, Auteri et al. [Auteri et al., 2013] applied the adjustment of the C/D ratio from PRISM to the Go-Go technique. In an object positioning task with time constraints, Go-Go+PRISM significantly improved the placement precision over the traditional Go-Go technique.

DOF separation is another common solution for mid-air manipulation. Separating the 6DOFs for translation and rotation can lead to better manipulation performance compared to controlling all 6DOFs simultaneously [Masliah et al., 2000]. The DS3 (Depth-Separated Screen Space) technique [Martinet et al., 2010] separated object positioning in depth and object positioning in screen space, which improved object

manipulation performance. If the initial position of the object is close to the users' hand, users could intersect the virtual hand/controller with axes attached/close to the object and manipulate the object by moving the axes [Mendes et al., 2017] [Caputo et al., 2018]. The axis-based techniques provided better accuracy for objects that were within arm's reach of the user. However, these techniques were not suitable for distant objects. Also, axis-based techniques require multiple selections and switches between different interaction modes, which reduces speed.

Some techniques used both hands for manipulation, e.g., the Handle Bar [Song et al., 2012] or Spindle+Wheel [Cho et al., 2015] techniques. However, the asymmetric bimanual gestures can increase the cognitive burden for the user. Also, the prolonged use of both hands in mid-air can result in additional fatigue, also due to the "gorilla arm" effect, a universal drawback shared by all mid-air interaction [Jang et al., 2017]. To address this issue, Hincapié-Ramos et al. [Hincapié-Ramos et al., 2014] suggested that 3D mid-air interaction techniques should limit hand motions to be between the hip and shoulders, which reduces the amount of arm extension and fatigue.

3D positioning techniques can be classified into constraint-based and free positioning ones. Many techniques used constraints to assist with object manipulation. With constraint-based positioning techniques, the position of the manipulated object is constrained by the attributes of other objects of the scene. Venolia presented "tail-dragging" [Venolia, 1993], where the user drags an object as it were attached to a rope. With a "snap-to" functionality, other objects also attract the manipulated object. Kitamura et al. proposed a "magnetic metaphor" for object manipulation [Kitamura et al., 2002], which aims to simulate physical behaviors, including non-penetration. Some multitouch techniques used constraints for 3D positioning, including snapping objects to another object [Au et al., 2012]. Building on object associations [Bukowski et al., 1995], Oh et al. [Oh et al., 2005] presented a sliding algorithm, where the object follows the cursor position directly and slides on any surface behind it, i.e., the moving object always stays attached to other objects. When dragging an object along a surface, users thus control two DOFs at any given time. The features of an extended sliding technique give the users control of the third DOF [Sun et al., 2016].

Currently, constraints are not widely used for 3D manipulation in VR. In Chapter 4, I implemented contact-plane sliding in VR using the Vive controller as input device. In

a 3D manipulation task with 2D devices, unconstrained manipulation is significantly slower [Smith et al., 2001]. However, whether this result holds true for 3D input devices is an open question.

3D selection and positioning techniques suffer from the Heisenberg effect [Bowman et al., 2001], where the slight movement of the wrist caused by a button press in the confirmation phase can affect the precision of object selection or placement. There are different approaches to reduce this effect. Vogel et al. [Vogel et al., 2005] proposed novel hand gestures to minimize such effects for mid-air selection. Techniques such as PRISM [Frees et al., 2005] scaled the C/D ratio between hand movement and object movement down, which mostly eliminates the Heisenberg effect. Another solution is bimanual interaction, where the non-dominant hand controls the mapping used during ray-casting [Kopper et al., 2008].

In this chapter, I adapt my extended sliding techniques to work in VR systems. Contact-plane sliding worked well in VR with the mouse as input device, as shown in my above-mentioned study. The performance of the extended sliding techniques in VR remains to be evaluated. In a hypothetical comparison of different input devices for extended sliding in VR, I hypothesize that the mouse would still perform better than the controller. The reason is that when using a controller for extended sliding, the three DOFs of object movement have to be separated: two DOF for contact-plane sliding, and one DOF for the extended features. When lifting an object off a surface, as in SHIFT-Sliding, some of DOFs of the controller's movement have to be disabled, which restricts the options for the user and takes some getting used to. Thus, my objective is to identify an appropriate mapping for extended sliding with a VR controller. In Section 5.2, I introduce the mappings for extended sliding in VR. In Section 5.3 and 5.4, I evaluate the performance of extended sliding by comparing it against different 3D positioning techniques.

5.2. Extended VR Sliding

Since the included controller is the most common input device for the HTC Vive HMD, I will use this VR controller as the input device and compare extended sliding against other 3DOF manipulation techniques. The Vive controller is a 6DOF input

device. In ray-casting based techniques, the translation and rotation of the controller both change the direction of the ray, and thus affect the position of the object. My goal is to determine if (extended) sliding is still a reliable positioning technique in VR with a controller, i.e., when a mouse is not available. I will not restrict head movements or rotations, as sliding is robust to these operations.

In Chapter 2, I extended contact-plane sliding to the third DOF. With SHIFT-Sliding, the users can move the object orthogonal to the sliding plane, forcing the object into a floating position. With DEPTH-POP, the users can move the object along the view direction. In Chapter 3, I used transparency techniques to extend sliding further to enable selection and positioning of hidden objects. In this chapter, I adapt SHIFT-Sliding to VR systems. DEPTH-POP and transparency features facilitate manipulation along the users' view direction, assuming camera navigation is time-consuming. However, in virtual environments with HMDs, users can easily move their head to see potentially hidden objects. Then users can select such objects with ray-casting selection. Therefore, DEPTH-POP and transparency features are relatively less beneficial in VR.

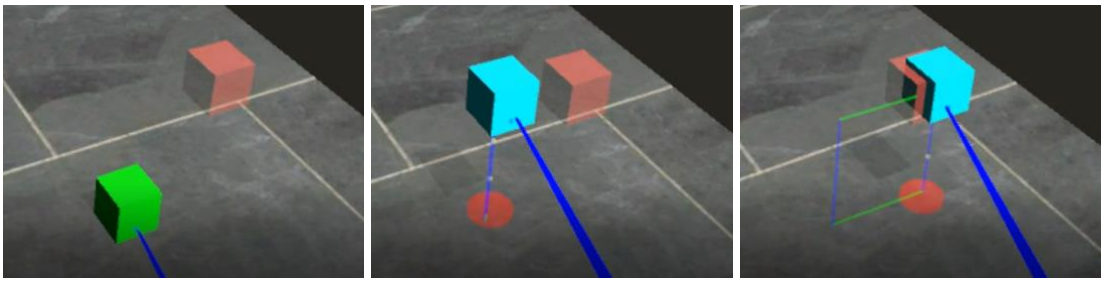


Figure 5.1. Illustration of Extended VR Sliding. The manipulated object is lifted up from the floor, while still being "under" the virtual cursor. The target position for the current trial is shown as the semi-transparent red cube, floating above the floor.

My implementation of contact-plane sliding in VR with the Vive controller was already introduced in Chapter 4. To adapt SHIFT-Sliding, I designed and implemented a prototype in which the action of the SHIFT key in the desktop system is mapped to pressing down the trackpad on the Vive controller. I call this technique *Extended VR Sliding*. Figure 5.1 illustrates Extended VR Sliding. A virtual sphere cursor follows the direction of the ray that originates at the controller and snaps onto the geometry that the ray intersects first. In the figure, the user previously pressed the trigger button to select

an object under the cursor and started sliding the object by moving the controller. When they also pressed the trackpad during sliding, further controller movement then lifts/pushes an object orthogonally to the current contact surface. When the object is lifted from a surface, I render it with a cyan color. Note that since I am mapping the 3DOF hand movement to 1DOF object movement orthogonal to the surface, it is impossible for the object to stay on a ray based on the surface normal direction *and* on the controller ray at the same time. Those two rays cannot intersect all the time. If I were to force the object to be along the normal direction regardless of the controller ray, this method would introduce an offset between the controller ray and object direction. Then, the only way to avoid the object "jumping" in an unexpected manner is to release the trigger and then re-select the object for parallel sliding. This method would require an additional selection task after each "lift", which would affect the efficiency of the technique.

To address this issue, I decided to fix object movement to a plane during lifting. When the trackpad is pressed during sliding, I construct a new plane based on the cross product of the normal vector and the controller ray direction. The object position is then defined by the intersection of that plane and the new controller ray. With this mapping, the object can be lifted from the surface, but still keeps its position under the controller ray. In essence, this means that lifting is then a 2DOF operation as up-down movements relative to the surface that the object was lifted from as well as some form of "sideways" movements are possible. As in the original SHIFT-Sliding technique, when the user releases the trackpad, they can keep moving the object parallel to the original contact surface in a continuous motion. Fernquist et al. introduced a snapping technique for touch interfaces that aligns objects to multiple constraints [Fernquist et al., 2011]. They designed a "catch-up" phase to overcome the offset between the object and the user's finger. However, If I were to apply this technique to the "lifting" phase of Extended VR Sliding, i.e., to snap the object along the normal direction and let it "catch-up" when the actual controller ray is farther off the normal direction, this would still not guarantee that the offset between the controller ray and object is resolved at all times when the trackpad is released. Extended VR Sliding has two stages: "lifting" and "sliding". If the user releases the trackpad in the "lifting" stage of user interaction while the system just enters the "catch-up" phase, this will bring a relatively large offset into the "sliding" stage, i.e., the object would not be under the controller ray anymore. Thus, I decided to still use

the solution discussed above, which keeps the object under the controller ray at all times.

If the trigger button is released during Extended VR Sliding, pressing the trigger again on the object will not snap the object back to a surface. Instead, the user can press the trigger to “re-capture” the object in the air and resume Extended VR Sliding. If there is no surface in the direction of the controller ray, pressing the trackpad during Extended VR Sliding moves the object orthogonal to the screen plane, therefore moving the object farther away/closer towards the user. When the trackpad is released, the user can then slide the object on a 2D (virtual) plane parallel to the screen plane.

In essence, Extended VR Sliding supports two different forms of 2DOF manipulation: one that slides the object on any surface and the second that lifts the object off the current surface. This separation into two 2DOF manipulation methods with a simple touch-based mode switch makes it easier to achieve accurate results, while reducing the effort and time for the mode change.

With Extended VR Sliding, I do not scale the C/D ratio from hand movement to object movement during manipulation. When the users release the trigger, the instability on their hand may cause the precision to drop [Bowman et al., 2001]. However, there is a trade-off between scaling the C/D ratio to increase precision and making sure that the object remains on the controller ray. A main design objective of my work is to always have the object remain on the controller ray without introducing any offset, which makes it easier for users to understand the technique. To reduce the potential negative effects introduced by releasing the trigger, I allow the user to use the trigger button on the other controller held in their non-dominant hand to release the object. According to the work of Batmaz et al. [Batmaz et al., 2019], using the dominant hand to point at objects and the non-dominant hand to select objects is more precise than using only a single controller in the dominant hand for a 3D selection task.

With ray-casting in VR, the user selects the object with the controller ray. In the normal ray-casting condition, the length of the ray is fixed, and the object stays at the end of the ray, which enables the user to manipulate all three DOFs for positioning but restricts the amount of changes in visual depth that can be achieved due to the limited length of the user’s arm. When a significant change of depth is required, it is more

beneficial to have a different mechanism to move the object in depth. An adapted version of the Ray-Casting-with-Reeling technique allows the user to adapt the length of the ray by pressing the top/bottom part of the trackpad on the Vive controller, similar to the ray length adjustment in the indirect HOMER technique [Bowman et al., 1997].

To evaluate my new technique and for positioning tasks where the initial and target positions of the object are substantially different in depth, I also implemented two virtual hand techniques that adjust the C/D ratio between the user's and the virtual hand: the Go-Go and the Go-Go+PRISM techniques. With both techniques, I use a 3D model of the controller as the representation of user's hand. I implemented a non-linear mapping of the virtual hand position so that the object and target positions are within reach if the users fully extend their hand. With Go-Go+PRISM, the C/D ratio between the user's and virtual hand also depends on the hand speed. Slowing down the hand movement will then increase the precision of 3D positioning.

5.3. User Study 1

I designed a user study that compared four techniques for 3D object positioning, Extended VR Sliding, as described above, Ray-Casting-with-Reeling, Go-Go, and Go-Go+PRISM. With Extended VR Sliding and Ray-Casting-with-Reeling, the user selects the object with the controller ray, as usual. In the Ray-Casting-with-Reeling condition, the object stays at the end of the ray, which enables the user to manipulate all three DOFs for positioning. The user is also able to change the length of the ray by pressing the top/bottom part of the trackpad on the Vive controller, similar to the ray length adjustment in the indirect HOMER technique [Bowman et al., 1997]. Figure 5.2 illustrates the Ray-Casting-with-Reeling condition. The length changes linearly with a constant speed.

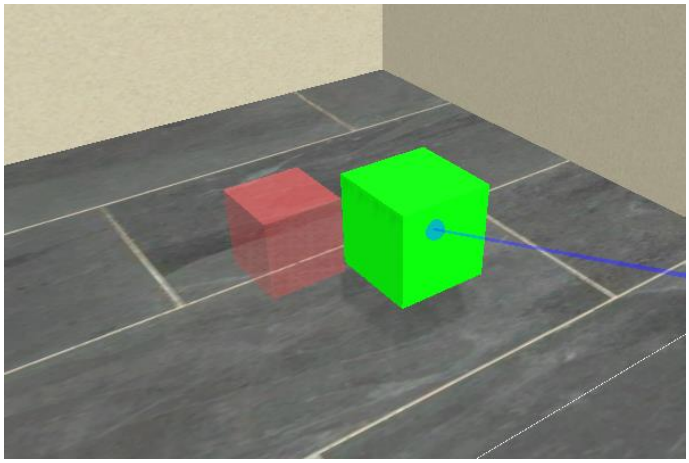


Figure 5.2. Illustration of Ray-Casting-with-Reeling. The object moves with the controller, while the length of the ray can be changed by pressing the trackpad.

In Go-Go and Go-Go+PRISM conditions, the user selects an object by intersecting a 3D model of the controller with the object. See Figure 5.3. I used a non-linear mapping between the controller position in the real world (also the user's hand) and the virtual controller position in the immersive environment. To implement this, I used the tracked headset position as a substitute for the user's torso position on the horizontal plane. If the distance between user's hand and torso is under a threshold (set to 10 cm in the study), the virtual controller position is the same as the real controller position. If the user's hand extends beyond the threshold, the virtual controller is further away along the direction of the controller's pointing direction. This implementation was different from the original Go-Go technique [Poupyrev et al., 1996], where they used the direction from the chest of the user to the user's hand. My implementation makes it easier to change the direction of the controller extension. When the target position is beside the initial object position and the required depth distance change is small, the users then only need to rotate the controller to move the object, as opposed to moving the controller laterally.

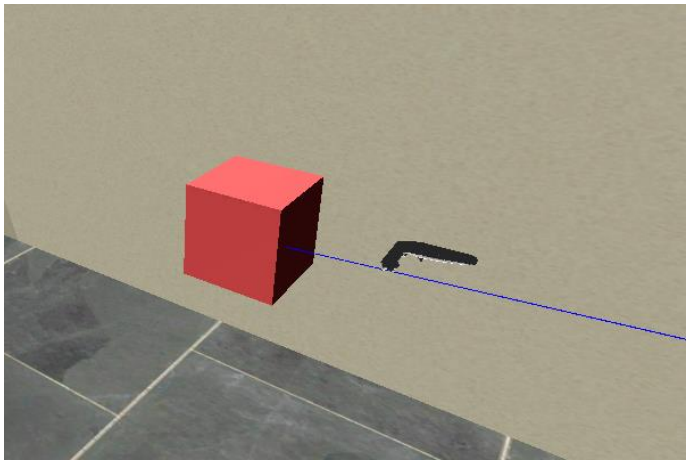


Figure 5.3. Illustration of Go-Go. The users intersect the virtual controller with the object for selection. The blue ray was not used for selection. It only shows the direction of the virtual controller extension.

$$D_v = \begin{cases} D_r, & D_r < thres \\ D_r + k * (D_v - D_r)^2, & D_r \geq thres \end{cases}$$

The equation above shows the computation of the virtual controller position. In the equation, D_v refers to the virtual controller's distance to user's torso. D_r refers to the real controller's distance to user's torso. If D_r is smaller than a threshold, the virtual controller position is the same as the real controller position. If D_r is greater than the threshold, D_v increases with a non-linear mapping. The constant k was set to 20.0 in my study. The virtual controller is then extended along the controller's pointing direction. The virtual controller's position is calculated by the intersection of the controller and an invisible reference cylinder, which is centered at user's head with a radius of D_r. The selected object will always follow the virtual controller position.

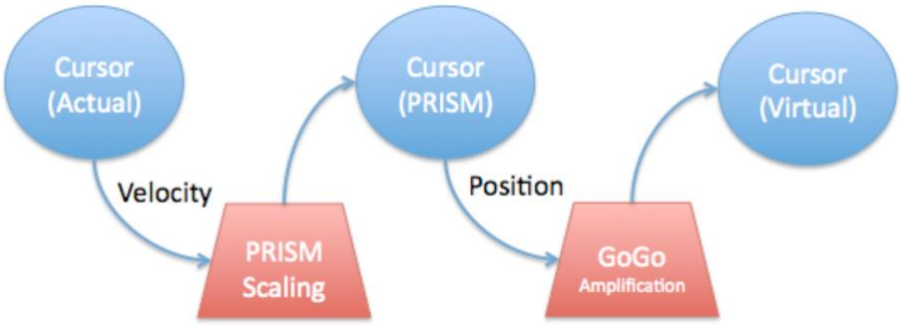


Figure 5.4. Cursor calculation workflow: The actual cursor velocity is used to control a PRISM controlled-virtual cursor, which is then amplified based on its position using Go-Go to provide the final interactive cursor.

In the Go-Go+PRISM condition, the actual controller velocity is used to control a PRISM controlled-virtual controller, which is then amplified based on its position using Go-Go to provide the final virtual controller position. Figure 5.4 (from [Auteri et al., 2013]) shows the cursor calculation workflow used in Go-Go+PRISM. If the users slow down the hand movement, it will slow down the controller movement, too. Moving the hand quickly, e.g., shaking the controller quickly, will reset any offset introduced to the controller by PRISM. Users naturally slow down the hand movement for fine adjustment if the object is close to the target. This potentially increase the precision of the positioning task.

If the users move their head in the Go-Go or Go-Go+PRISM condition, it changes the virtual controller position, therefore changing the object position. Thus, I told the users to try to keep their head (reasonably) still during the experiment. Also, moving the hand forward/backward at the side of the body is not the most efficient way of changing the virtual controller's position. Therefore, I instructed the users to try to move their hand closer/further away from their body.

To avoid the Heisenberg effect [Bowman et al., 2001] in all four techniques, the users press the trigger button on the controller in their non-dominant hand for confirmation. The means that users do not have to release the trigger on the dominant hand to finalize the object position and I hypothesize that this would effectively reduce the Heisenberg effect.

5.3.1. Apparatus & Participants

The hardware setup for the experiment used a first generation HTC Vive (headset, base stations, controllers). The Vive HMD has a diagonal field of view of 110 degrees, a display resolution of 1080x1200, and a refresh rate of 90 Hz. I used a desktop computer with 3.6 GHz i7 processor, 16 GB of memory, a NVIDIA GeForce GTX 980 graphics card, and Unity 2018 for the implementation. I used the desktop monitor to observe the users' actions during the experiments.

I recruited 12 (1 female) unpaid students from the local university population. I did not screen participants for 3D/VR experience. Six of the participants played games regularly. I had ethics approval for the user study, and participants signed consent forms. All the participants were informed about the potential risk of motion sickness.

5.3.2. Procedure

I designed a 3D object positioning experiment and asked participants to move an object from a starting to a target position as quickly and as accurately as possible in various scenes. The target position was rendered as semi-transparent. The users pressed the trigger button on their dominant hand to select the object. When the user positioned the object at the target position, they had to press the trigger button on the Vive controller in their non-dominant hand to confirm the position and to go to the next trial.

There was a 3-minute training session before each condition, which introduced participants to the techniques in a playground environment that did not include any version of the experimental tasks. After the users finished all the tasks, I asked them to fill a questionnaire about the usability of the four techniques. In total, the study took about 45 minutes for each participant.

5.3.3. Experimental Design

I designed a user study to evaluate the performance of Extended VR Sliding. The study focuses on 3D positioning of distant objects, i.e., when the initial position of the

object is far away from the user. The participants sit in front of a table, wearing the Vive headset. They hold the Vive controllers in both hands. I calibrated the virtual room to match the real one to some degree, so that the users are more comfortable when transitioning to using the system in a sitting position. The virtual room is five meters on each side. Figure 5.5 shows one of the task scenes. The user sits in the center of the virtual room, which makes it impossible for them to reach the walls of the room with their arms. In all trials, the object is a cube with 20 cm side length. The starting and target positions are at least two meters away from the user in all trials, therefore not directly reachable by the controller, i.e., out of reach of the user's hand. Still, except for moving the controller with their arm it was possible to finish all trials without significant body movement.

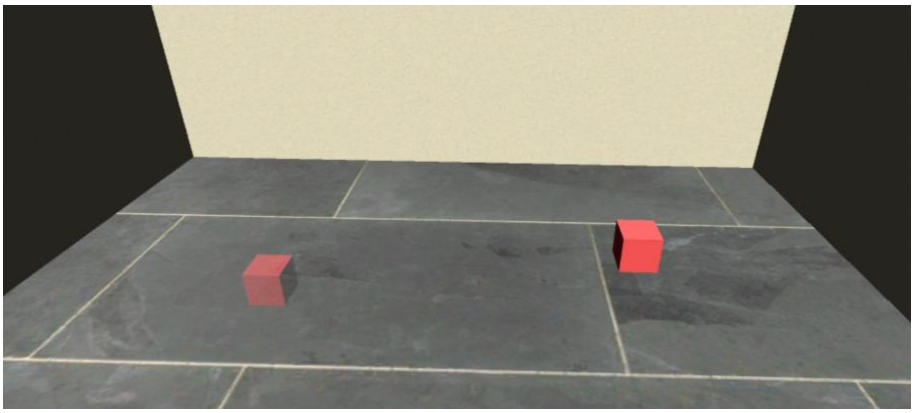


Figure 5.5. Illustration of one task scene. The object is rendered opaque in red. The target position is rendered semi-transparently. The target is in contact with the floor. The depth distance change is small.

In the user study, I asked participants to move the object from a starting to a target position in various scenes. When the users position the object at the target position, they press the trigger button on the Vive controller in their non-dominant hand to confirm the final position and to go to the next trial. There were two major dependent variables: positioning time and error distance. I measured the positioning time in seconds and the error distance in centimeters. The timing started when the object was first selected and ended when the user released the trigger button for the last time. The error distance is the 3D distance between the object's center and the ideal target position's center. The error measure was calculated as the absolute distance between

the object's center and the ideal target position's center over the object size (20 cm). I recorded all actions of each user.

Finally, after they finished all the tasks, I asked the users to complete a questionnaire, where they rated the ease of use, perceived speed, and fatigue of each technique. I used a Likert scale with seven levels. For ease of use, 1 means very difficult to use, 7 means very easy, and 4 means neutral. For perceived speed, 7 means very fast, 1 means very slow, and 4 means neutral. For fatigue level, 7 means very low, 1 means very high, and 4 means neutral. I also asked participants about their preference among each pair of techniques. Participants also provided feedback on the techniques and the experiment.

The experiment had a 4 (technique) x 2 (depth distance change) x 2 (object contact) within-subject design, where I used a balanced Latin square for technique to counterbalance the order of the tasks to reduce learning effects. A balanced Latin square with size 4 has 4 different orders [Mackenzie, 2012]. Since I recruited 12 participants, each bin had 3 participants in it. For depth distance change, I randomly assigned half of the participants to perform the small condition first, the other performed the large condition first. For object contact, half of the participants performed the contact condition first, the other half performed the floating condition first. In each sub-condition, there were 5 trials. Those 5 trials were not repetitions. Instead, users had to move the object to different positions, where the target positions matched the requirement of the specific sub-condition. Therefore, each participant had to perform 80 trials. My study included three independent variables: technique, depth distance change, and object contact, as follows. I compared three previously presented **techniques** with Extended VR Sliding: Ray-Casting-with-Reeling, Go-Go, and Go-Go+PRISM. All four techniques can handle tasks where the initial object position is far away from the user, and also when the target position is far away from the initial object position. Extended VR Sliding and Ray-Casting-with-Reeling represent techniques that use ray-casting for object selection, which Go-Go and Go-Go+PRISM use a virtual. The last two techniques use a scaled C/D ratio, and Go-Go+PRISM combines Go-Go with PRISM to enable the user to achieve higher precision.

Besides the main effect of technique, I also wanted to analyze if different scenes influence user performance. To study this aspect, I included conditions with two levels of

depth distance change: small and large. When the change in object depth is small, normally a small change in arm extension is required with previously presented techniques, and vice versa. However, Extended VR Sliding does not necessarily require a large arm extension when the depth distance change is large. Thus, I wanted to study how different depth changes affect the performance of 3D positioning tasks. In the small condition, the target position will be relatively close to the initial position in terms of target depth. In such tasks, the depth distance change will be less than 50 cm. Except for moving the controller with their arm, all trials could then be finished without significant body movement. In the large condition, the depth distance change will be more than 250 cm, which will require larger body movements, which should result in longer execution times.

The third independent variable, **object contact**, has two levels: contact and floating. I hypothesize that object contact will significantly impact (and improve) the performance of Extended VR Sliding, but not the other techniques. In the contact condition, all target positions are in contact with the rest of the scene. All tasks can then be finished with the contact-plane sliding technique. If the target position of the object is in contact or close to a surface in the scene, this will improve the efficiency and accuracy of Extended VR Sliding. In the floating condition, all the target positions will be at least 50 cm away from the closest surface in the scene.

In this study, I measured positioning time and positioning accuracy. I hypothesize that the two virtual hand techniques will require more hand movement in depth than the ray-casting based techniques, therefore causing more fatigue (H1). Since half of the tasks include targets in contact, I also hypothesize that Extended VR Sliding will be as accurate as Go-Go+PRISM, and both will be significantly more accurate than Ray-Casting-with-Reeling or Go-Go (H2). Further, I hypothesize that positioning time will be significantly lower for tasks with a small depth distance change, as less hand movement in depth is required for virtual hand techniques (H3). Finally, I hypothesize that Extended VR Sliding will be significantly faster and more accurate for targets that are in contact with the scene, while object contact will not have a significant effect on the performance of the other techniques (H4).

5.3.4. User Study Results

Shapiro Wilk tests were conducted to assess the normality of the dataset. The results showed that the data of neither positioning time nor error measure was normally distributed ($p < .0001$). However, for positioning time and error measure, the skewness and kurtosis of the data were between -2 and +2. This made it acceptable to prove normal univariate distribution [George, 2011]. Therefore, despite the violations, I decided to move on for further ANOVA tests.

ANOVA Results on Positioning Time

I performed a 3-way (4 Technique x 2 Object Contact x 2 Depth Distance Change) repeated measures ANOVA on positioning time. The ANOVA results showed that the main effect of Technique on positioning time was significant, $F(3, 33) = 10.56$, $p < .0001$, $\eta_{p2} = 0.05$. A Tukey post-hoc analysis showed that Go-Go ($M = 3.89$s, $SD = 2.04$s) was significantly faster than Extended VR Sliding ($M = 5.89$s, $SD = 4.66$s) and Ray-Casting-with-Reeling ($M = 5.74$s, $SD = 4.03$s). Go-Go+PRISM ($M = 4.92$s, $SD = 3.80$s) did not have any significant difference between the other three techniques. See Figure 5.6. In this figure and all the following data plots in this chapter, I used * ($p < .05$), ** ($p < 0.01$), *** ($p < .001$) to represent the Tukey post-hoc pairwise comparison between any two conditions that turned out to be significant. Any comparisons that I did not report were not significantly different.

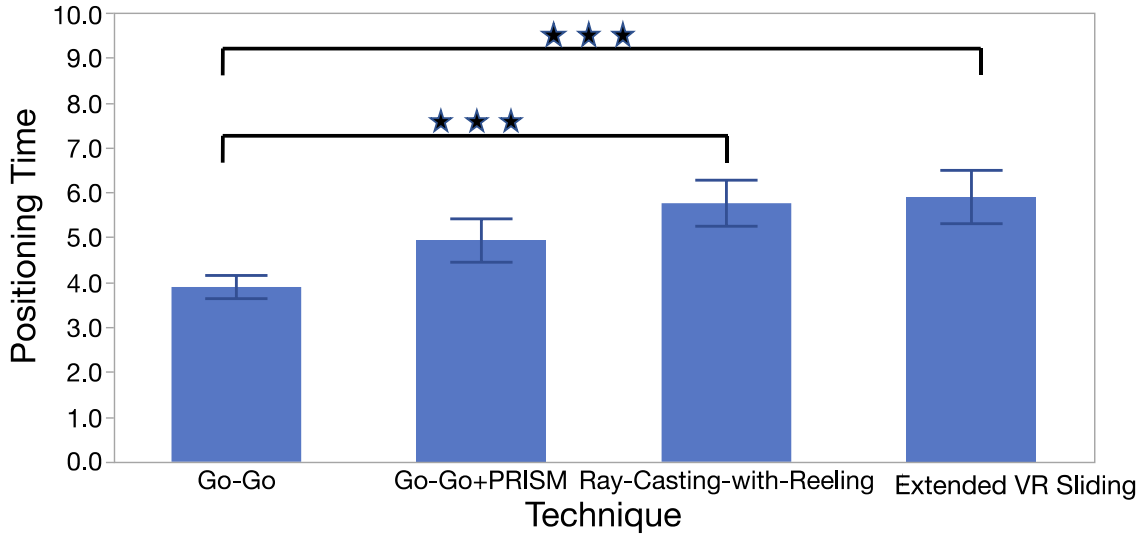


Figure 5.6. Average positioning times (in seconds) for techniques. Each error bar is constructed using a 95% confidence interval of the mean.

The ANOVA results also showed that the main effect of Object Contact on positioning time was significant, $F(1, 11) = 11.44$, $p = .0061$, $\eta_{p2} = 0.005$. The Contact condition ($M = 4.86$s, $SD = 3.58$s) was significantly faster than the Floating condition ($M = 5.37$s, $SD = 4.07$s).

The ANOVA results also identified that the main effect of Depth Distance Change on positioning time was not significant, $F(1, 11) = 1.36$, $p = .2678$, $\eta_{p2} = 0.001$. The Small condition ($M = 4.99$s, $SD = 3.89$s) and the Large condition ($M = 5.24$s, $SD = 3.78$s) did not yield a significant difference on the positioning time. This did not match my hypothesis **H3**.

The ANOVA results also identified that the Technique x Object Contact interaction had a significant effect on positioning time, $F(3, 33) = 36.84$, $p < .0001$, $\eta_{p2} = 0.133$. Extended VR Sliding, Contact ($M = 3.31$s) was the fastest combination. This matched my hypothesis **H4**. However, Extended VR Sliding, Floating ($M = 8.46$s) was the slowest combination. See Figure 5.7.

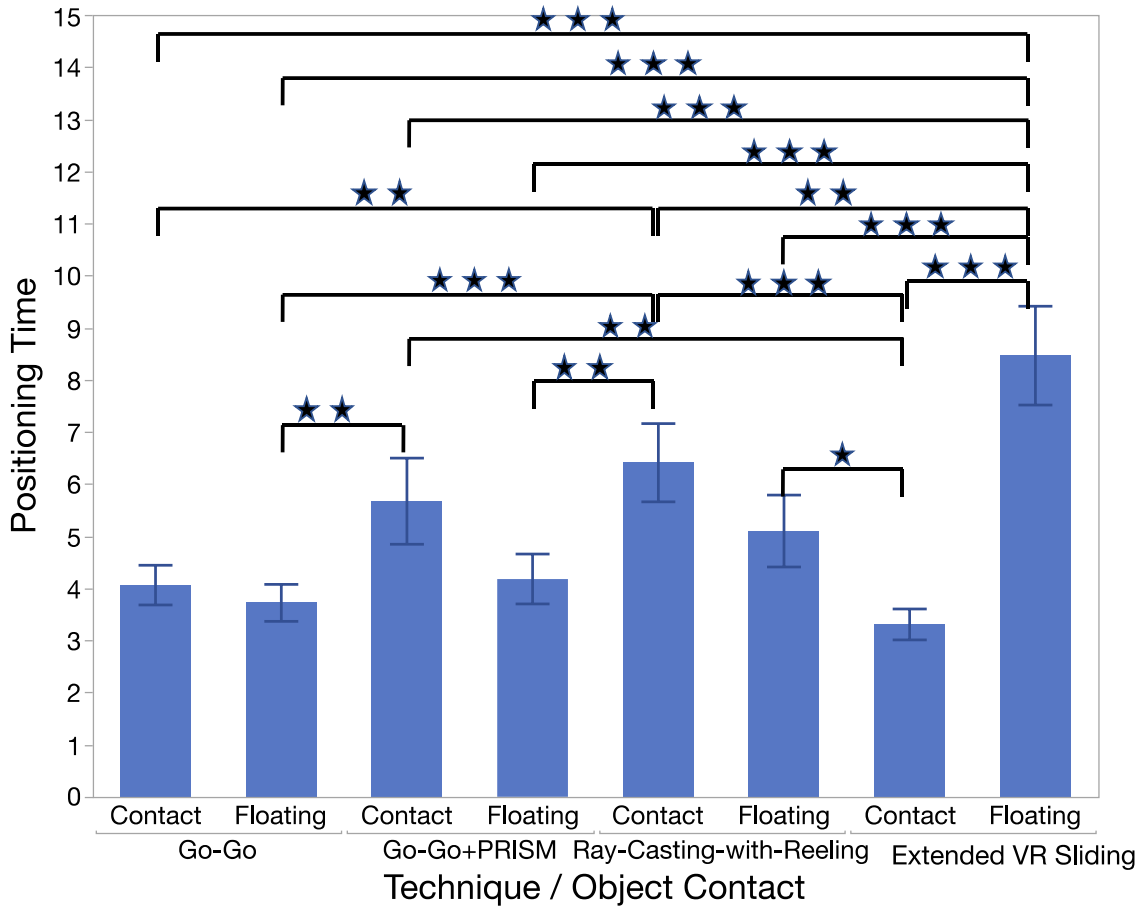


Figure 5.7. Average positioning times for technique and object contact. Each error bar is constructed using a 95% confidence interval of the mean.

The ANOVA results also showed that the Technique x Depth Distance Change interaction had a significant effect on positioning time, $F(3, 33) = 4.02$, $p = .0154$, $\eta_{p2} = 0.007$. Go-Go, Small ($M = 3.82$s) and Go-Go, Large ($M = 3.96$s) were the fastest combinations. Ray-Casting-with-Reeling, Large ($M = 6.34$s) was the slowest combination. For the two virtual hand-based techniques, depth distance change did not have a significant impact on positioning time. This further denied my hypothesis **H3**. See Figure 5.8.

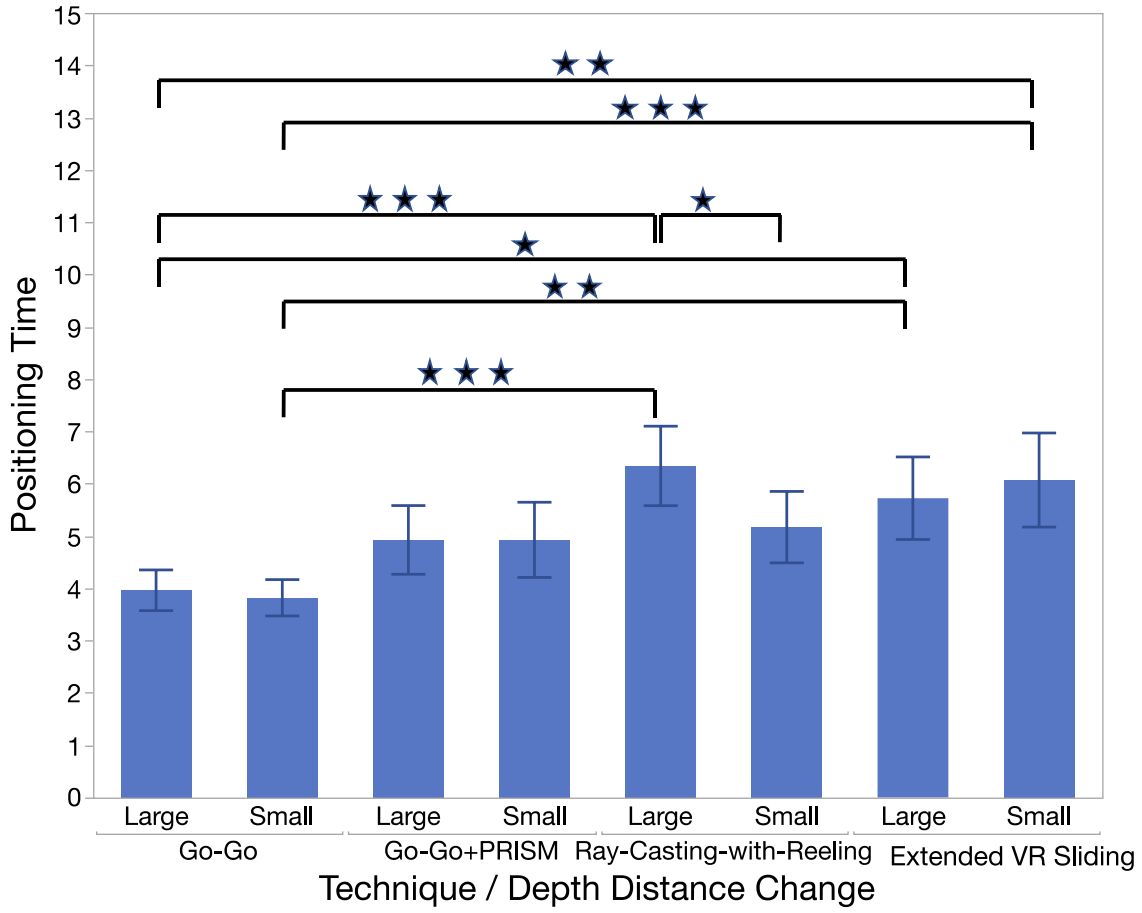


Figure 5.8. Average positioning times for technique and depth distance change. Each error bar is constructed using a 95% confidence interval of the mean.

ANOVA Results on Error Measure

In terms of error measure, I performed a 3-way (4 Technique x 2 Object Contact x 2 Depth Distance Change) repeated measures ANOVA. The ANOVA results showed that the main effect of Technique on error measure was not significant, $F(3, 33) = 2.15$, $p = .1125$, $\eta_{p2} = 0.011$. A Tukey post-hoc analysis showed that Extended VR Sliding ($M = 0.417$, $SD = 0.520$), Go-Go ($M = 0.435$, $SD = 0.452$), Ray-Casting-with-Reeling ($M = 0.508$, $SD = 0.511$), and Go-Go+PRISM ($M = 0.545$, $SD = 0.552$) did not have a significant difference in error measure. This partly supported my hypothesis **H2**. Extended VR Sliding was not significantly different from any other techniques for error measure. See Figure 5.9.

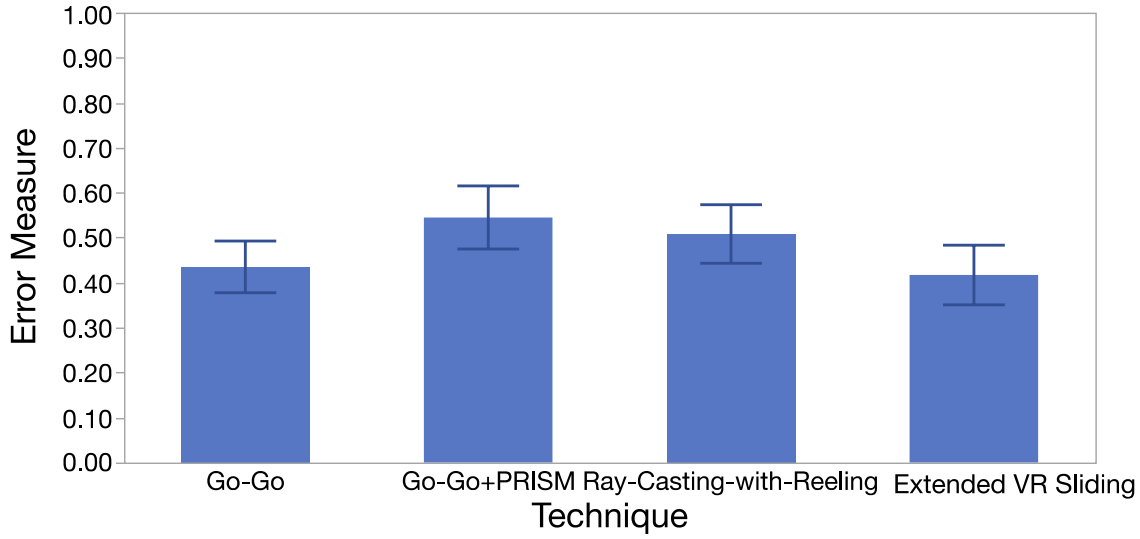


Figure 5.9. Average error measures for techniques. Each error bar is constructed using a 95% confidence interval of the mean.

The ANOVA results also showed that the main effect of Object Contact on error measure was significant, $F(1, 11) = 6.94$, $p = .0232$, $\eta_{p2} = 0.036$. The Contact condition ($M = 0.382$, $SD = 0.466$) was significantly more accurate than the Floating condition ($M = 0.570$, $SD = 0.538$).

The ANOVA results also identified that the main effect of Depth Distance Change on error measure was significant, $F(1, 11) = 10.13$, $p = .0087$, $\eta_{p2} = 0.005$. The Small condition ($M = 0.442$, $SD = 0.476$) was significantly more accurate than the Large condition ($M = 0.510$, $SD = 0.544$).

The ANOVA results also identified that the Technique x Object Contact interaction had a significant effect on the error measure, $F(3, 33) = 7.75$, $p = .0005$, $\eta_{p2} = 0.035$. Extended VR Sliding, Contact ($M = 0.159$) was the most accurate combination. This again confirmed my hypothesis **H4**. See Figure 5.10.

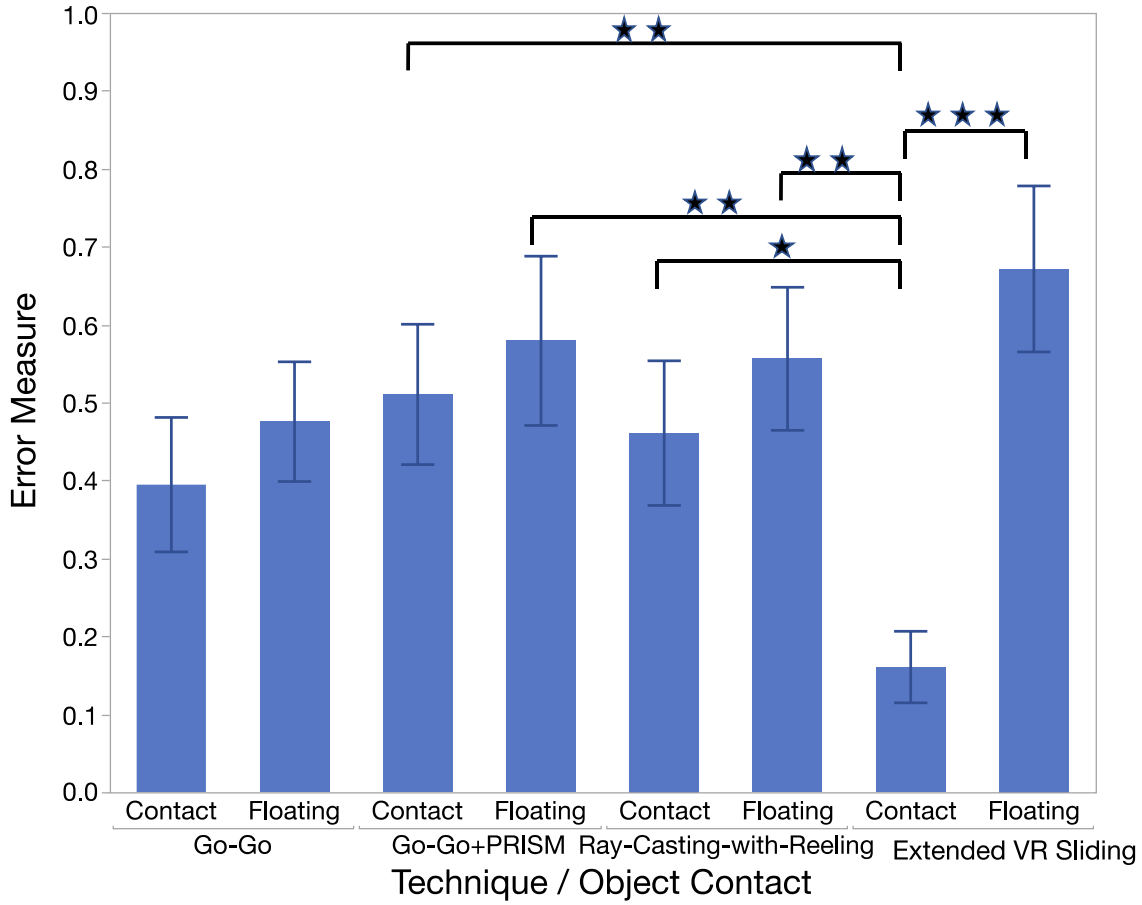


Figure 5.10. Average error measures for technique and object contact. Each error bar is constructed using a 95% confidence interval of the mean.

The ANOVA results also identified that the Technique x Depth Distance Change interaction did not have a significant effect on error measure, $F(3, 33) = 0.54$, $p = .6555$, $\eta_{p2} = 0.001$. See Figure 5.11.

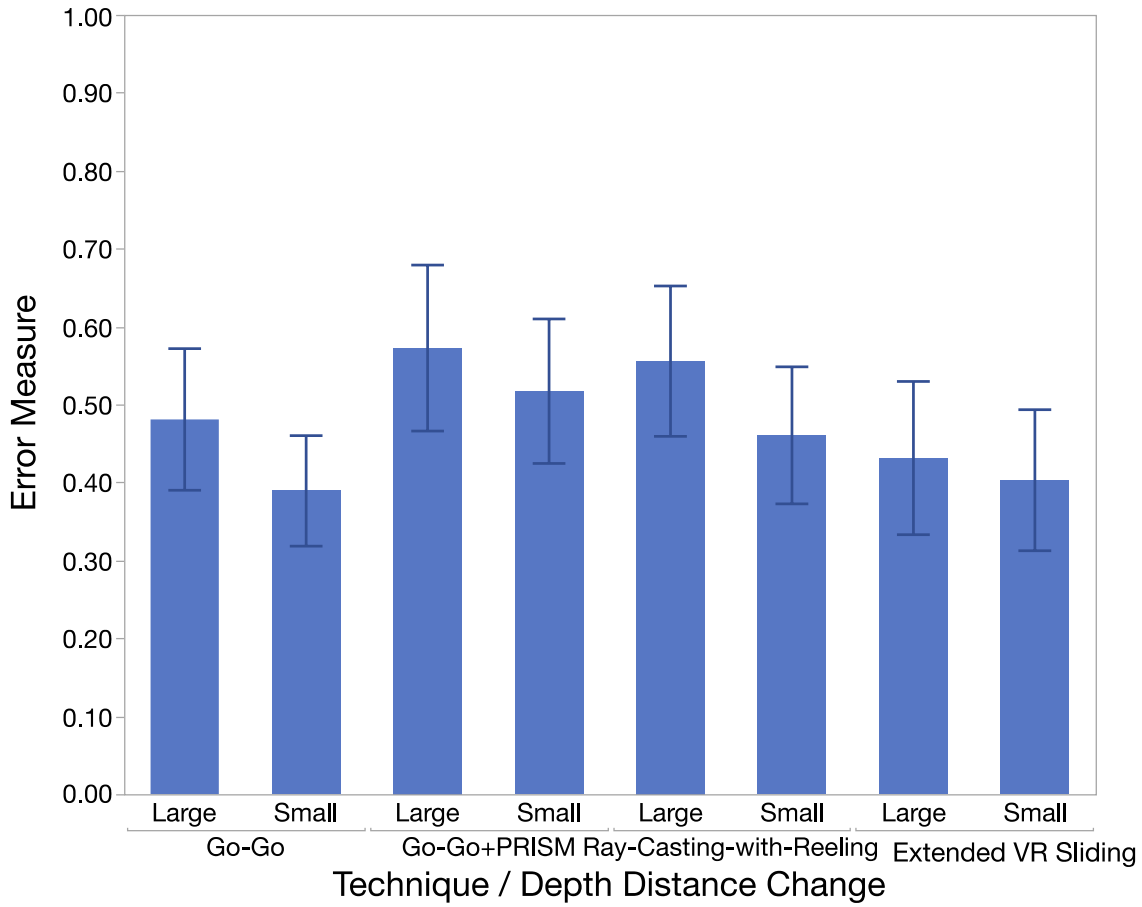


Figure 5.11. Average error measures for technique and depth distance change. Each error bar is constructed using a 95% confidence interval of the mean.

Questionnaire Responses

Seven out of 12 participants found Extended VR Sliding neutral or easy to use. Four participants found Go-Go+PRISM neutral or easy to use. Ten participants found Go-Go and Ray-Casting-with-Reeling neutral or easy to use. See Figure 5.12. Previous studies had shown that it is feasible to analyze Likert scale data with parametric statistics, even with small sample sizes, unequal variances, and non-normal distribution [Carifio et al., 2008] [Norman, 2010]. Therefore, I performed repeated measures ANOVA on the Likert scale data. The ANOVA results showed that technique had a significant effect on ease of interaction, $F(3, 33) = 19.92$, $p < .0001$, $\eta_{p2} = 0.239$. Go-Go ($M = 5.50$) and Ray-Casting-with-Reeling ($M = 5.00$) were significantly easier to use than Extended VR Sliding ($M = 3.83$) and Go-Go+PRISM ($M = 3.67$).

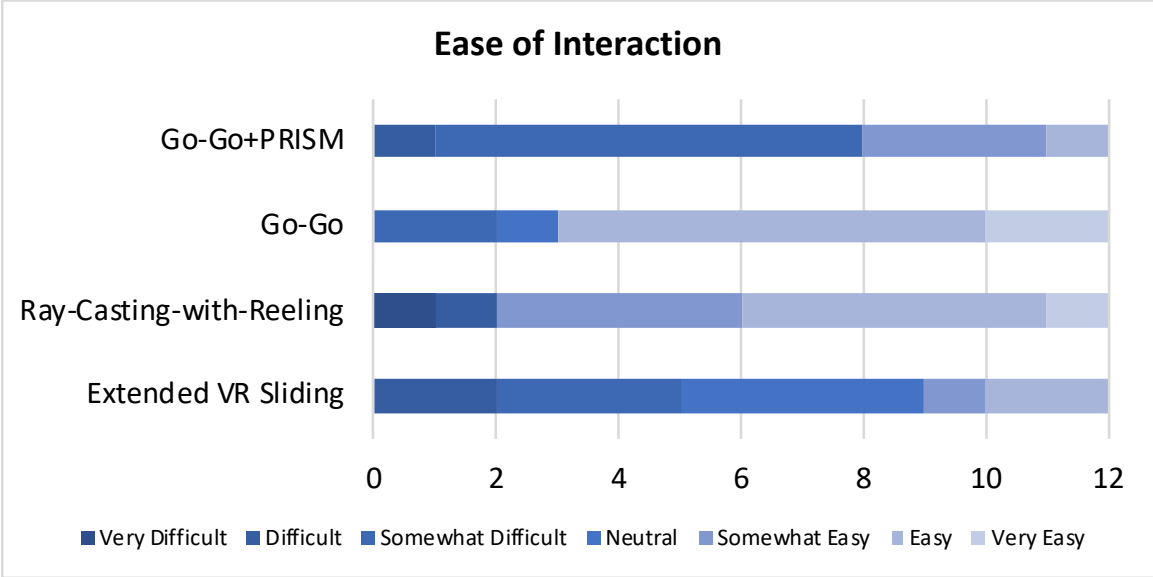


Figure 5.12. Ease of interaction responses from all participants.

Nine out of 12 participants found the perceived speed of Extended VR Sliding natural or fast. Seven participants found the perceived speed of Go-Go+PRISM and Ray-Casting-with-Reeling neutral or fast. Ten participants found the perceived speed of Go-Go neutral of fast. See Figure 5.13. The ANOVA results showed that technique had a significant effect on perceived speed of interaction, $F(3, 33) = 12.25$, $p < .0001$, $\eta_{p2} = 0.087$. Go-Go ($M = 5.25$) was perceived to be significantly faster than Ray-Casting-with-Reeling ($M = 4.33$), Extended VR Sliding ($M = 4.25$) and Go-Go+PRISM ($M = 4.25$).

Eight out of 12 participants found the fatigue level of Extended VR Sliding neutral or low. Nine found the fatigue level of Go-Go and Ray-casting-with-Reeling neutral or low. Four found the fatigue level of Go-Go+PRISM neutral or low. The ANOVA results showed that technique had a significant effect on the fatigue level of interaction, $F(3, 33) = 25.43$, $p < .0001$, $\eta_{p2} = 0.127$. Ray-Casting-with-Reeling ($M = 4.83$) was the least fatiguing, while Go-Go+PRISM ($M = 3.67$) was most fatiguing. Go-Go ($M = 4.25$) and Extended VR Sliding ($M = 4.42$) were not significantly different. This partly supported my hypothesis **H1**. Go-Go+PRISM, one of the virtual hand-based techniques, was indeed the most fatiguing. Extended VR Sliding was only more fatiguing than Ray-Casting-with-Reeling. See Figure 5.14.

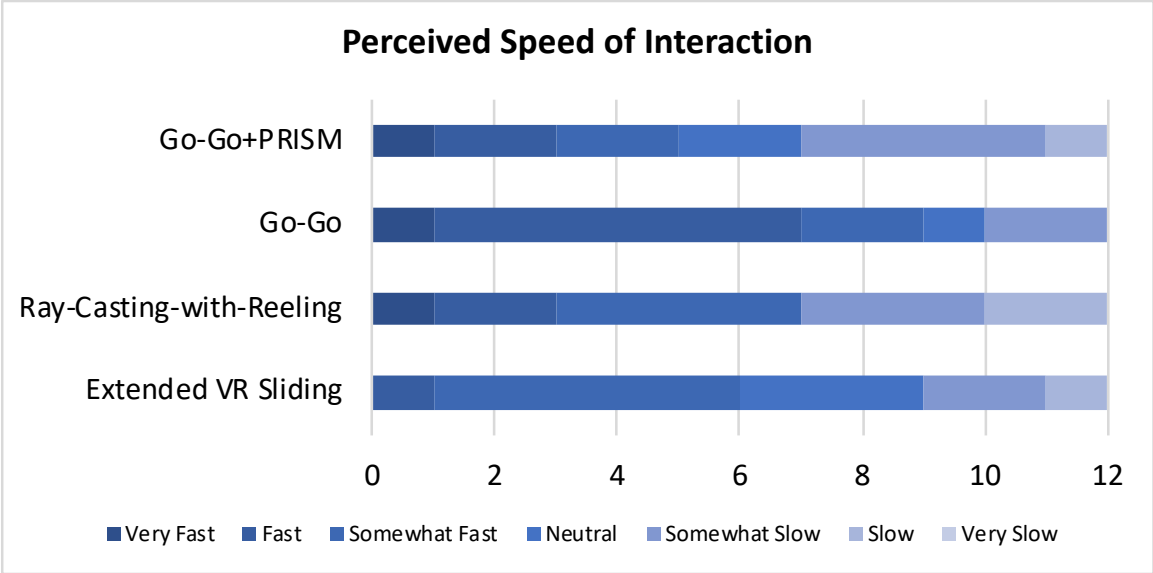


Figure 5.13. Perceived speed of interaction responses from all participants.

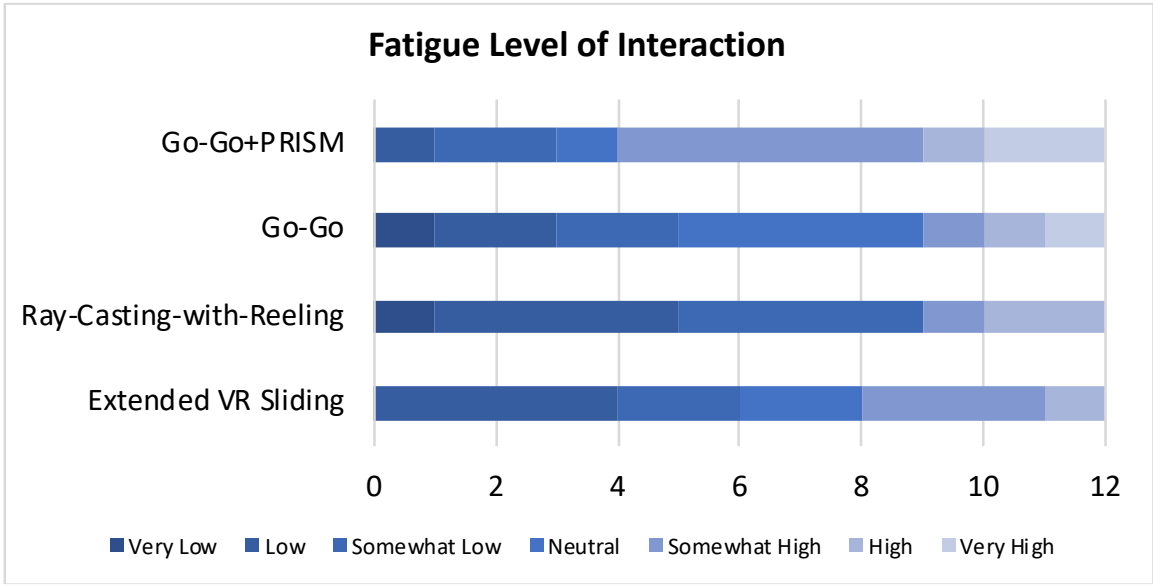


Figure 5.14. Fatigue level of interaction responses from all participants.

Six participants rated Go-Go as their favourite technique. Three participants rated Extended VR Sliding as their favourite. Three rated Ray-Casting-with-Reeling as their favourite. Eleven participants rated Go-Go over Go-Go+PRISM.

I also asked the participants to provide comments or suggestions on the techniques. Three participants identified that the ray-casting based techniques, i.e., Extended VR Sliding and Ray-Casting-with-Reeling, were better for object selection than

virtual hand-based techniques, i.e., Go-Go and Go-Go+PRISM. Six participants rated Go-Go as their favourite, as they claimed it was natural to use. Only four participants rated Go-Go+PRISM neutral or easy to use. Four participants commented that since Go-Go+PRISM mapped slower hand movement to object movement, they found it difficult to having to reset the virtual controller position when an offset was introduced unintentionally. They found Go-Go+PRISM too slow and physically uncomfortable to use.

Five participants found the mappings used by Extended VR Sliding unintuitive to use or perceived the mode change to be challenging. Yet, two participants commented that when the target was in contact with the scene, Extended VR Sliding was the easiest among the four techniques. Three participants commented that they did not like Ray-Casting-with-Reeling since they had to press the trackpad on the Vive controller while holding the trigger button. Interestingly, the other two participants liked Ray-Casting-with-Reeling for the same reason, as they found it easier to control the depth of the object with the trackpad button than moving their hand forward/backward.

5.4. Simplified Extended VR Sliding

Based on the results of user study 1, and especially the user feedback, I decided to make some modification to the Extended VR Sliding technique to improve its usability.

One common feedback I got about the Extended VR Sliding technique was that the mappings were not natural or intuitive enough. In my original implementation for Extended VR Sliding, I made a design decision that the object always stays on the controller ray. Some users found this unintuitive, as moving the hand forward and backwards did not change the direction of controller ray much, which therefore did not change the object's position in depth, as some participants seem to expect. Still, most users managed to learn that with Extended VR Sliding they had to change the orientation of the controller to slide the object in the training phase.

If the object was floating above a surface, users had to press down on the trackpad to lift the object off the surface. This lifting action afforded technically only 2DOF, yet some users perceived it more to be like free hand manipulation. When the user released the trackpad, the 2DOF lifting phase was finished, and the system

transitioned to a mode that affords 2DOF sliding parallel to the surface that the object was previously in contact with. Then, the object movement was again not much determined by controller position, but instead by the ray determined by the controller orientation. Some users did not understand this transition well and found it hard to position an object with precision, and therefore did not perform well in the condition where the target object was floating.

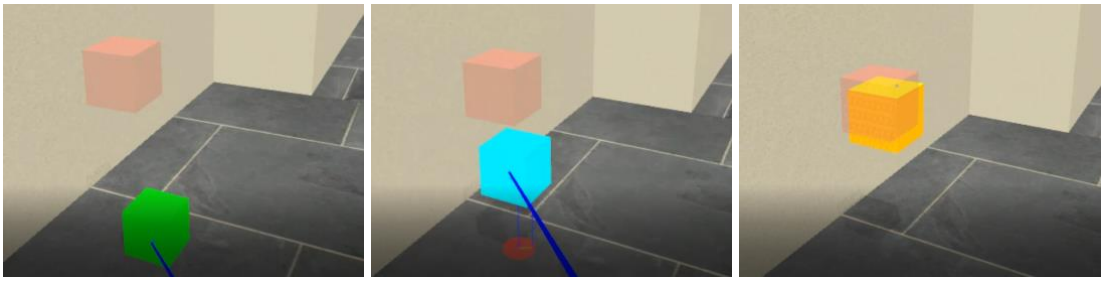


Figure 5.15. Illustration of the free hand adjustment phase in the simplified Extended VR Sliding technique. To visualize the mode change, the ray is not shown anymore and the object is shown in yellow. The user then moves the object with controller movement through a scaled virtual hand mapping.

Although I believe even more training could help users understand Extended VR Sliding better, I decided to adjust the technique to be easier to understand. I kept the sliding and lifting phases but removed the final parallel sliding phase. Instead, after the users release the trackpad, I make the controller ray disappear. The users can now move the object with their (virtual) 3D hand movement. To illustrate the mode change, I rendered the object with a yellow color in this phase. See Figure 5.12. The user's hand (controller) movement is then directly mapped to object movement, magnified by a constant (set to 5.0 in experiment). Overall, the sliding and lifting phases are then used to move the object close to the target position, and the fine adjustment phase can be achieved by free 3D hand movements. I hypothesize that this change would address the users' comments and make the technique more natural and easier to use. For the confirmation of the placement, I still used the trigger on the user's non-dominant hand.

5.5. User Study 2

I designed another user study to evaluate the performance of the simplified Extended VR Sliding technique.

I included Go-Go+PRISM in user study 1 because it might improve the precision for the 3D positioning tasks. However, the results of user study 1 demonstrated otherwise. Go-Go+PRISM was both not more accurate and also significantly slower than Go-Go. Moreover, 11 out of 12 participants preferred Go-Go over Go-Go+PRISM. I think the reason for this is that users did not like the offset between the virtual hand and object positions introduced by Go-Go+PRISM. This offset was supposed to ease the Heisenberg effect, as I expected the users to slow down their hand movement for fine adjustment when the target was close. However, some users slowed their hand movement away from the target and were surprised by the introduced offset. Then, they had to reset the controller movement. Besides, I used the trigger button on the non-dominant hand for confirmation, which also reduced the Heisenberg effect. Therefore, I decided to remove the Go-Go+PRISM condition from this study.

Ray-Casting-with-Reeling was significantly slower than Go-Go. For the interaction of technique and depth distance change, Ray-Casting-with-Reeling, Large was the slowest combination. Ray-Casting-with-Reeling was also not significantly more accurate than any other technique. Therefore, I decided to remove the Ray-Casting-with-Reeling condition. Also, I made the depth distance change even larger in user study 2, so any associated effects would show up more clearly.

5.5.1. Apparatus & Participants

The hardware setup for the experiment was the same as user study 1. I again recruited 12 (5 female) new unpaid students from the local university population, who had not participated in user study 1. I did not screen participants for 3D/VR experience. Four of the participants played games regularly. I had ethics approval for the user study, and participants signed consent forms. All the participants were informed about the potential risk of motion sickness.

5.5.2. Procedure

The procedure of the experiment was similar to user study 1 and involved a 3D object positioning task that asked participants to move an object from a starting to a target position as quickly and as accurately as possible in various scenes. The target position was rendered as semi-transparent. The users pressed the trigger button on their dominant hand to select the object. When the user positioned the object at the target position, they had to press the trigger button on the Vive controller in their non-dominant hand to confirm the placement and go to the next trial.

There was a 3-minute training session before each condition, which introduced participants to the techniques in a playground environment that did not include any version of the experimental tasks. After the users finished all the tasks, I asked them to fill a questionnaire about the usability of all techniques. Since there were only two techniques, the study took about 30 minutes for each participant.

5.5.3. Experimental Design

In this experiment, I compared the performance of the simplified Extended VR Sliding technique and Go-Go. I used a similar setup as user study 1. This study focuses on 3D positioning of distant objects, i.e., when the initial position of the object is far away from the user. The participants sat in front of a table, wearing the Vive headset. They held a Vive controller in each hand. To increase the depth change, I made the virtual room ten meters in depth, which was deeper than the one in user study 1. See Figure 5.13. In all trials, the object was a cube with 30 cm side length. The starting and target positions were least two meters away from the user in all trials, well out of reach. I set the constant k in Go-Go to be 40.0 so that all trials could be finished by moving the controller with their arm. I recorded the same measurements as user study 1. I used the same questionnaire as user study 1.

Figure 5.16. Illustration of one task scene. The object (far) is rendered opaque in red. The target position (near) is rendered semi-transparently, here on the right and floats above the floor. The depth distance change in this trial is large.

The experiment had a 2 (technique) x 2 (depth distance change) x 2 (object contact) within-subject design, where I used a balanced Latin square to counterbalance the order of the tasks to reduce learning effects. For each independent variable, half of the participants performed one condition first and the other half performed the other half first. In each sub-condition, there were 5 trials. Those 5 trials were not repetitions. Instead, users had to move the object to different positions, where the target positions matched the requirement of the specific sub-condition. Therefore, each participant had to perform 40 trials. My study included three independent variables: technique, depth distance change, and object contact, as follows. I compared two **techniques**: Extended VR Sliding and Go-Go. Both techniques can handle tasks where the initial object position is far away from the users and the target position is far away from the initial object position in terms of (visual) depth.

I included two levels of **depth distance change**: small and large. When the change in object depth is small, normally a small change in arm extension is required, and vice versa. In the small condition, the target position is relatively close to the initial position in terms of target depth. In such tasks, the depth distance change is less than 1 m. In the large condition, the depth distance change was more than 5 m, which requires larger body movements during manipulation, which should result in longer execution times. Except for moving the controller with their arm, users were still able to finish all trials without significant body movement.

The third independent variable, **object contact**, has two levels: contact and floating. In the contact condition, all the target positions are in contact with the rest of the scene. All tasks can then be finished with the contact-plane sliding technique. In the floating condition, all the target positions will be at least 1 m away from the closest scene surface.

I measured positioning time and positioning accuracy. I hypothesize that Extended VR Sliding will be more accurate than Go-Go, as shown by user study 1 (H1). Also, due to the modifications I introduced, I hypothesize that Extended VR Sliding will be as fast as Go-Go (H2). Finally, I hypothesize that Extended VR Sliding will be significantly faster and more accurate for targets that are in contact with the scene, while object contact will not have a significant effect on Go-Go (H3).

5.5.4. User Study Results

To assess the normality of the dataset, I conducted Shapiro Wilk tests. The results showed that the data of neither positioning time nor error measure was normally distributed ($p < .0001$). However, for positioning time and error measure, the skewness and kurtosis of the data were between -2 and +2. This made it acceptable to prove normal univariate distribution [George, 2011]. Therefore, despite the violations, I decided to move on for further ANOVA tests.

ANOVA Results on Positioning Time

I performed a 3-way (2 Technique x 2 Object Contact x 2 Depth Distance Change) repeated measures ANOVA on positioning time. The ANOVA results showed that the main effect of Technique on positioning time was significant, $F(1, 11) = 19.32$, $p = .0011$, $\eta_{p2} = 0.033$. Go-Go ($M = 4.41$s, $SD = 2.37$s) was significantly faster than Extended VR Sliding ($M = 5.20$s, $SD = 2.76$s). This did not match my hypothesis **H2**. See Figure 5.17.

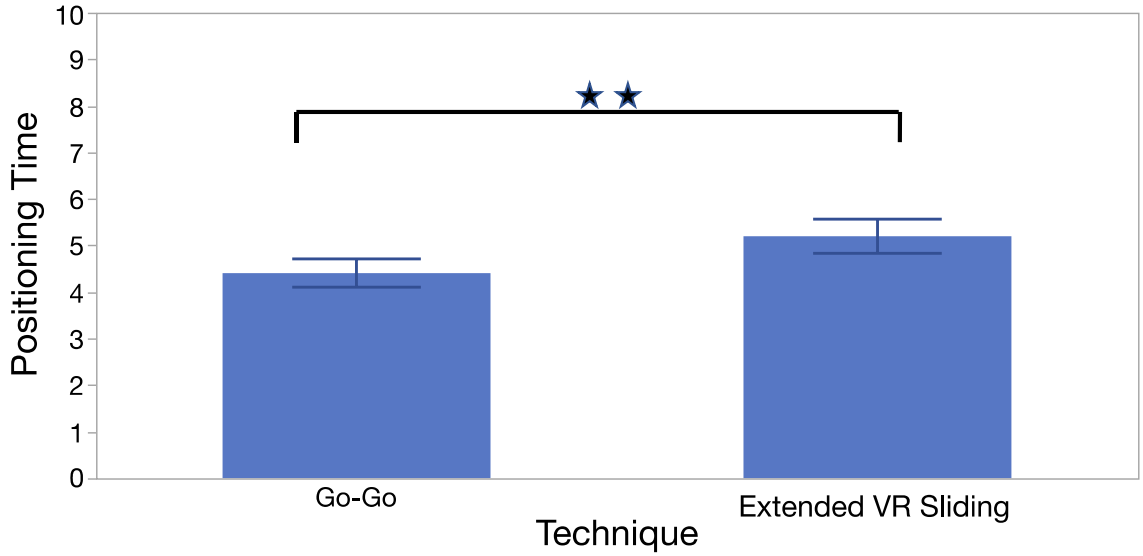


Figure 5.17. Average positioning times for technique. Each error bar is constructed using a 95% confidence interval of the mean.

The ANOVA results also identified that the main effect of Object Contact on positioning time was significant, $F(1, 11) = 24.13$, $p = .0005$, $\eta_{p2} = 0.069$. The Contact condition ($M = 4.21$s, $SD = 2.29$s) was significantly faster than the Floating condition ($M = 5.39$s, $SD = 2.74$s).

The ANOVA results also showed that the main effect of Depth Distance Change on positioning time was not significant, $F(1, 11) = 3.68$, $p = .0795$, $\eta_{p2} = 0.003$. The Small condition ($M = 4.67$s, $SD = 2.56$s) and the Large condition ($M = 4.90$s, $SD = 2.62$s) did not yield a significant difference on the positioning time.

The ANOVA results also identified that the Technique x Object Contact interaction had a significant effect on positioning time, $F(1, 11) = 145.89$, $p < .0001$, $\eta_{p2} = 0.123$. Extended VR Sliding, Contact ($M = 3.77$s) was the fastest combination. This matched my hypothesis **H3**. However, Extended VR Sliding, Floating ($M = 6.76$s) was the slowest combination. See Figure 5.18.

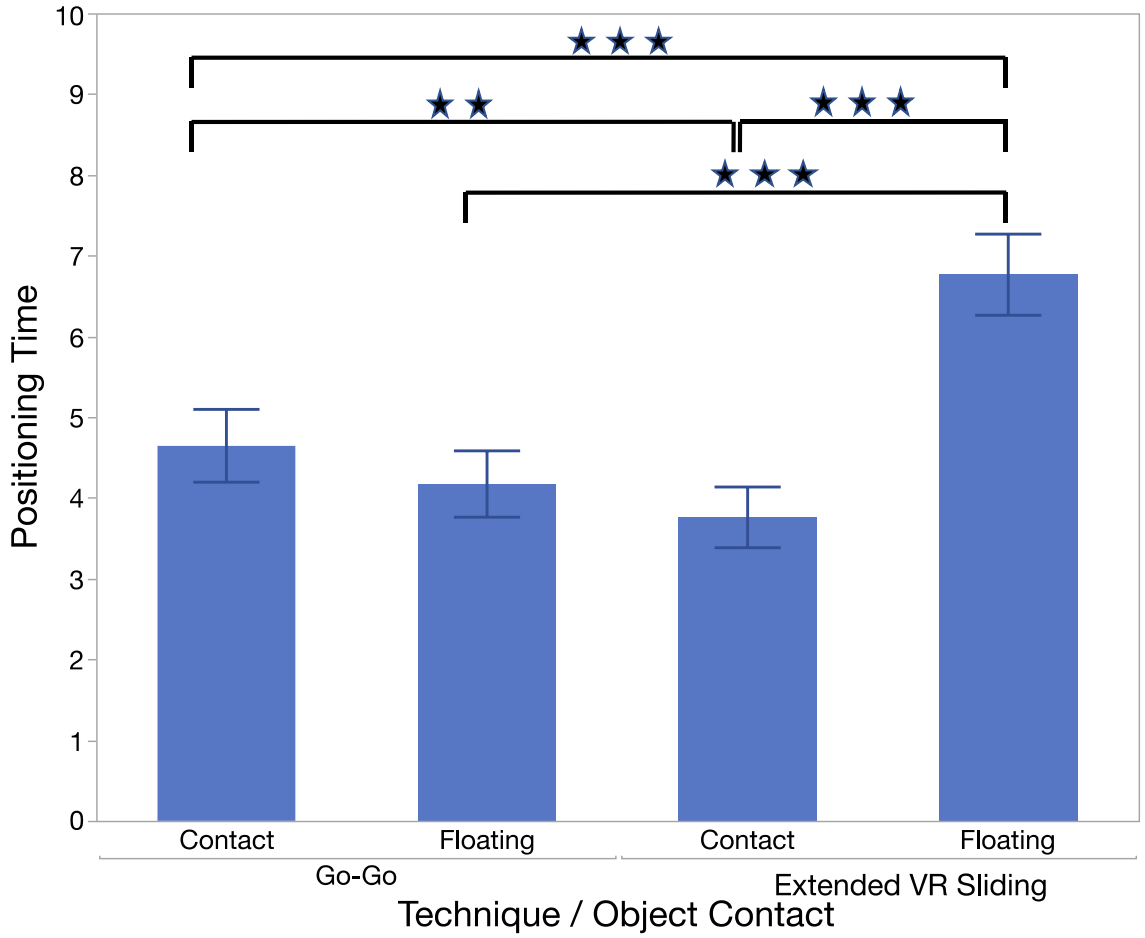


Figure 5.18. Average positioning times for technique and object contact. Each error bar is constructed using a 95% confidence interval of the mean.

The ANOVA results identified that the Technique x Depth Distance Change interaction did not have a significant effect on positioning time, $F(1, 11) = 0.62$, $p = .4500$, $\eta_{p2} = 0.001$. However, a Tukey post-hoc analysis showed that Go-Go, Small ($M = 4.20$s) was the fastest combination. See Figure 5.19.

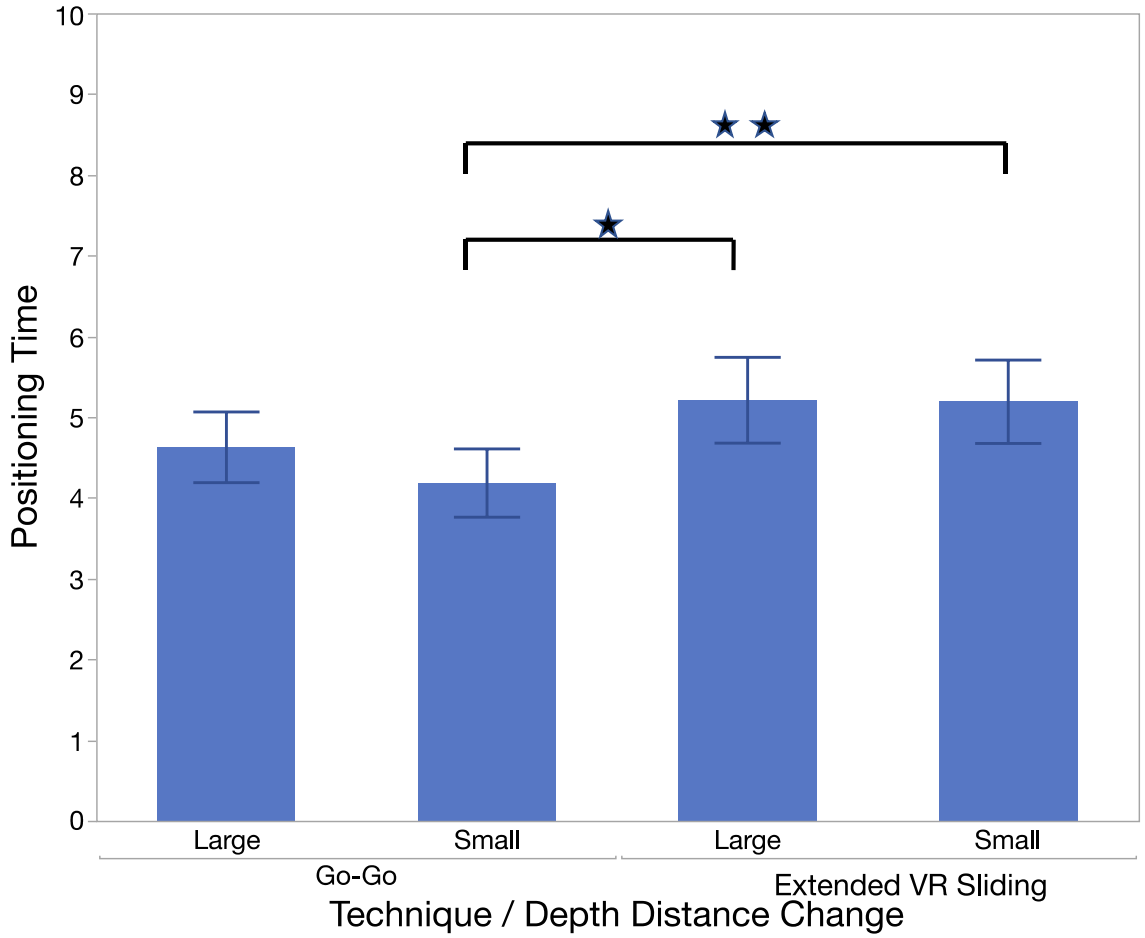


Figure 5.19. Average positioning times for technique and depth distance change. Each error bar is constructed using a 95% confidence interval of the mean.

ANOVA Results on Error Measure

In terms of error measure, I performed a 3-way (2 Techniques x 2 Object Contact x 2 Depth Distance Change) repeated measures ANOVA. The ANOVA results showed that the main effect of Technique on error measure was significant, $F(1, 11) = 10.60$, $p = .0079$, $\eta_{p2} = 0.016$. Extended VR Sliding ($M = 0.255$, $SD = 0.321$) was significantly more accurate than Go-Go ($M = 0.344$, $SD = 0.357$). This matched my hypothesis **H1**. See Figure 5.20.

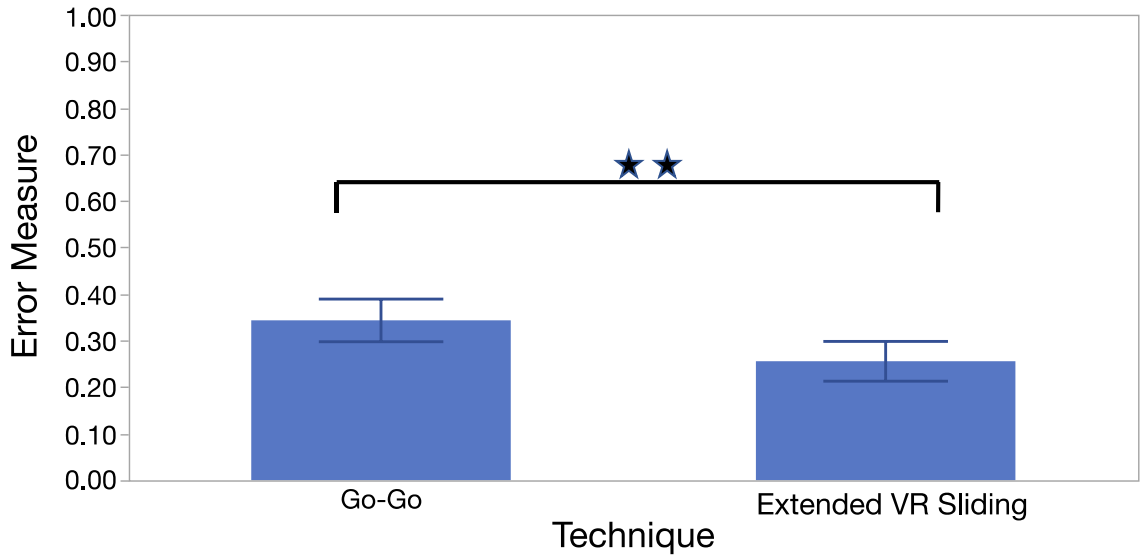


Figure 5.20. Average error measures for technique. Each error bar is constructed using a 95% confidence interval of the mean.

The ANOVA results also showed that the main effect of Object Contact on error measure was not significant, $F(1, 11) = 4.32$, $p = .0614$, $\eta_{p2} = 0.018$. The Contact condition ($M = 0.256$, $SD = 0.273$) and the Floating condition ($M = 0.347$, $SD = 0.396$) did not yield a significant difference on the error measure.

The main effect of Depth Distance Change on error measure was not significant, $F(1, 11) = 3.60$, $p = .0914$, $\eta_{p2} = 0.003$. The Small condition ($M = 0.282$, $SD = 0.320$) and the Large condition ($M = 0.320$, $SD = 0.363$) did not yield a significant difference on the error measure.

The ANOVA results also showed that the Technique x Object Contact interaction did not have a significant effect on error measure, $F(1, 11) = 0.95$, $p = .3492$, $\eta_{p2} = 0.001$. However, a Tukey post-hoc analysis showed that Extended VR Sliding, Contact ($M = 0.201$) was significantly more accurate than Go-Go, Contact ($M = 0.305$) and Go-Go, Floating ($M = 0.381$). This further confirmed my hypothesis **H3**. See Figure 5.21.

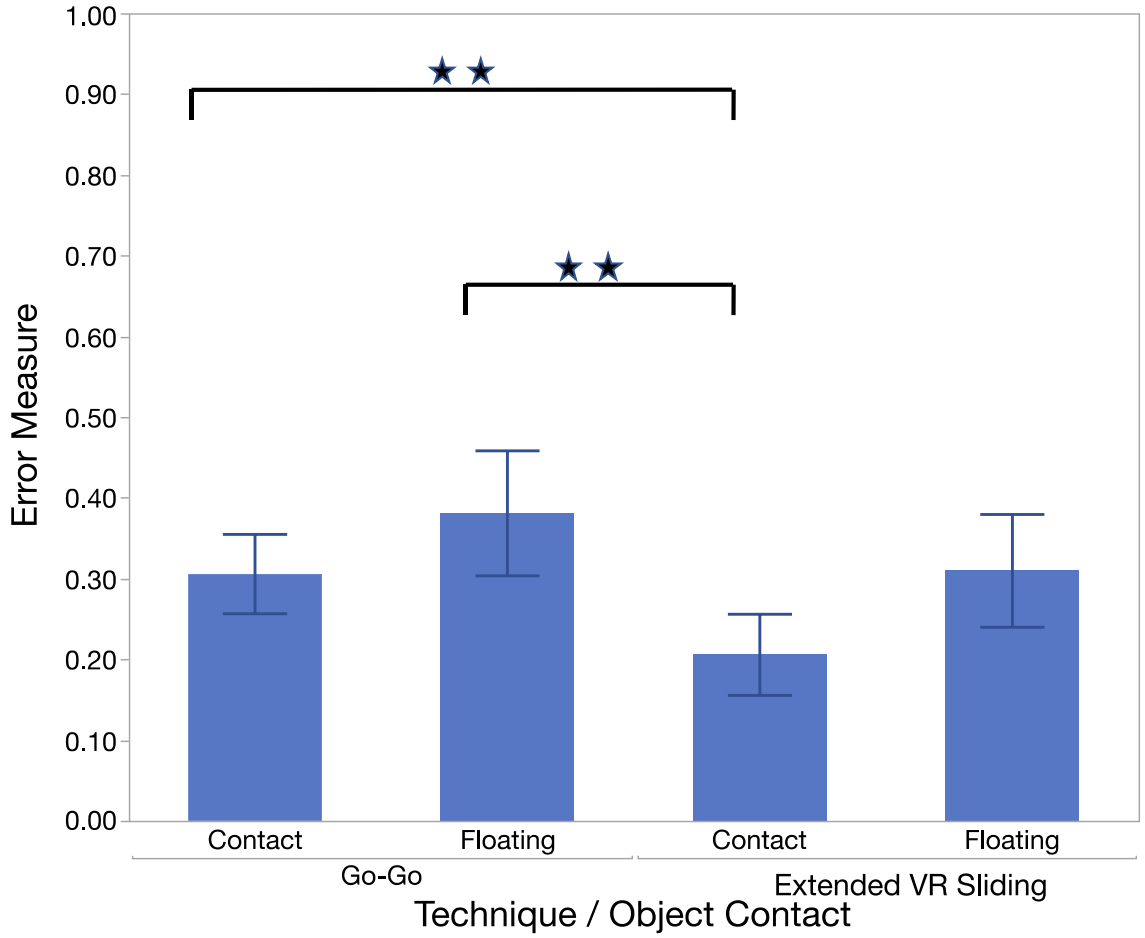


Figure 5.21. Average error measures for technique and object contact. Each error bar is constructed using a 95% confidence interval of the mean.

The ANOVA results also identified that the Technique x Depth Distance Change interaction did not have a significant effect on error measure, $F(1, 11) = 1.00$, $p = .3403$, $\eta_{p2} = 0.002$. However, a Tukey post-hoc analysis showed that Go-Go, Large ($M = 0.373$) was the least accurate combination. See Figure 5.22.

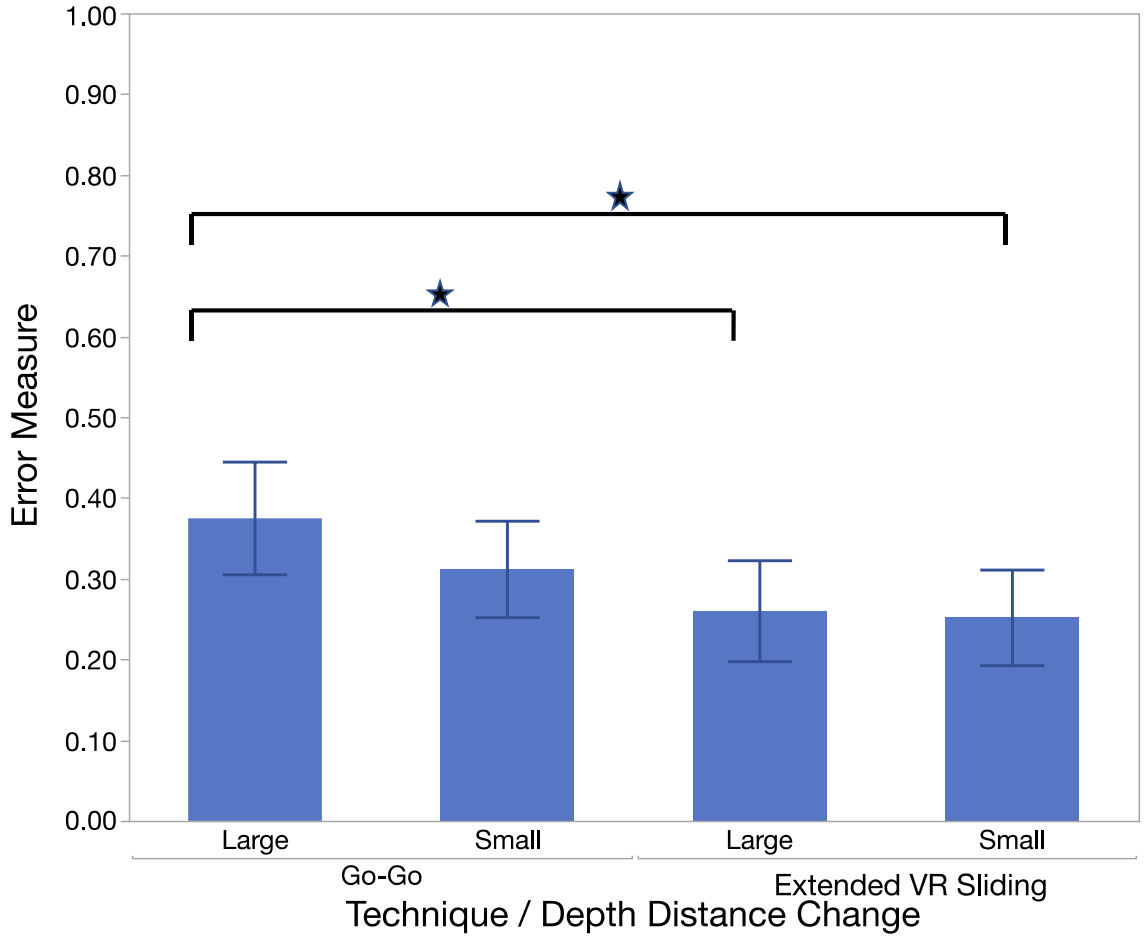


Figure 5.22. Average error measures for technique and depth distance change. Each error bar is constructed using a 95% confidence interval of the mean.

Questionnaire Responses

Ten out of 12 participants found Extended VR Sliding neutral or easy to use. Eleven participants found Go-Go neutral or easy to use. See Figure 5.23. The repeated measures ANOVA results showed that technique had a significant effect on ease of interaction, $F(1, 11) = 6.60$, $p = .0261$, $\eta_{p2} = 0.054$. Go-Go ($M = 5.33$) was easier to use than Extended VR Sliding ($M = 4.83$).

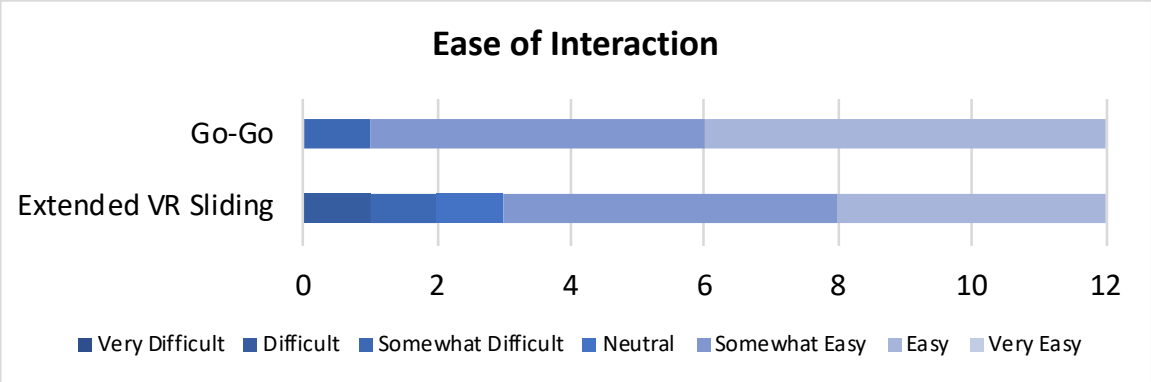


Figure 5.23. Ease of interaction responses from all participants.

Eight participants found the perceived speed of Extended VR Sliding to be neutral or fast. Eleven found the perceived speed of Go-Go neutral or fast. See Figure 5.24. The ANOVA results showed that technique had a significant effect on the perceived speed of interaction, $F(1, 11) = 16.18$, $p = .0020$, $\eta_{p2} = 0.131$. Go-Go ($M = 5.00$) was perceived to be faster than Extended VR Sliding ($M = 4.17$).

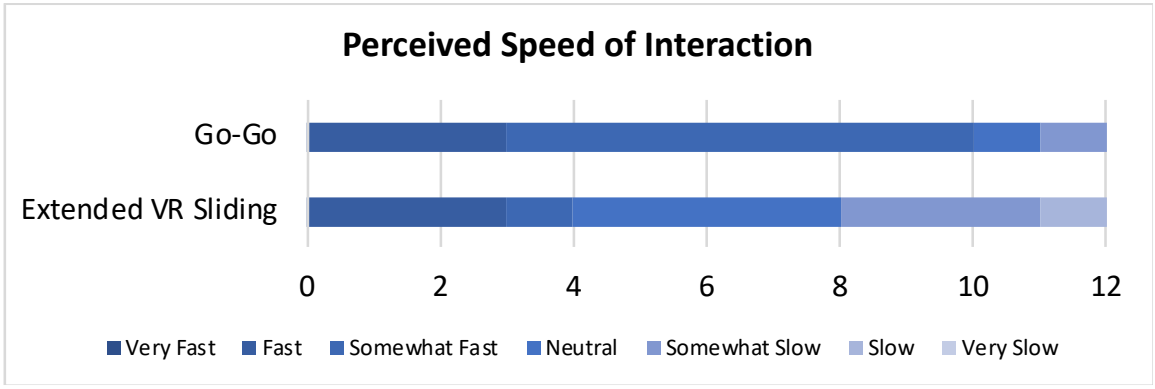


Figure 5.24. Perceived speed of interaction responses from all participants.

Eleven out of 12 participants found the fatigue level of Extended VR Sliding to be neutral or low. Nine found the fatigue level of Go-Go neutral or low. See Figure 5.25. The ANOVA results showed that technique had a significant effect on fatigue level of interaction, $F(1, 11) = 11.96$, $p = .0054$, $\eta_{p2} = 0.088$. Extended VR Sliding ($M = 5.25$) was less fatiguing than Go-Go ($M = 4.42$).

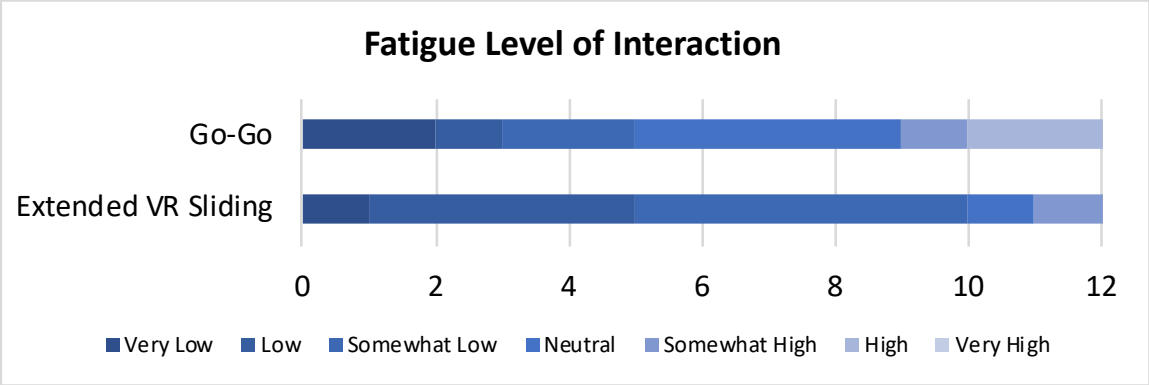


Figure 5.25. Fatigue level of interaction responses from all participants.

Moreover, seven participants preferred Extended VR Sliding over Go-Go. One was neutral.

I asked the participants to provide comments or suggestions on the techniques. Three participants rated the fatigue level of Go-Go above neutral, as in some situations when the depth distance change was large, the technique required users to stretch their arm fully to reach an object. Five participants expressed a preference for the modified version of Extended VR Sliding, i.e., using free hand movement after lifting an object. This modification helped participants to be more precise in the fine adjustment phase and they found the mode change to be easy and natural. When the object had already been slid close to the target and the floating distance was relatively small, I observed that at least three participants used the trackpad merely as a button for mode change, i.e., forsook the lifting phase and went directly to the free hand mode.

Additional Results

I recorded all actions of the users. Based on the records, I could analyze users' behaviour in more depth. The average positioning time for the Extended VR Sliding, Contact combination was 3.77s. The average positioning time for the Extended VR Sliding, Floating combination was 6.76s. The difference between these two combinations was due to the two extra phases required to position floating objects. The sliding phase in the floating conditions took 3.12s on average, which was faster than the contact condition, as no fine adjustment was required. The lifting phase took 1.47s on average. After releasing the trackpad, the users spent 2.17s on average for free hand fine adjustments.

Compared to user study 1, the average error measure for the Extended VR Sliding, Floating combination dropped from 0.671 (user study 1) to 0.310 (user study 2). I combined the data for the Extended VR Sliding, Floating condition in both studies and tagged each data point with the corresponding study number (1 and 2). I then performed a t-test of Study Number on error measure. The result showed a significant difference between study 1 (M = 0.671, SD = 0.589) and study 2 (M = 0.310, SD = 0.361), $p <$ .0001. The modification of Extended VR Sliding to support free hand fine adjustment significantly improved the accuracy for the floating condition. As the depth distance change is set to be different between two studies, I did not have a between-study analysis of positioning time.

5.6. Discussion

The results of user study 1 showed that Extended VR Sliding was as accurate, yet not as fast as other free hand techniques. This was not unexpected, as Extended VR Sliding constrains object movement to 2DOF. Users could thus not manipulate all 3 DOFs simultaneously. Also, for floating objects, users had to switch between three different types of 2DOF movements, which may cause a cognitive burden to them. In comparison, Go-Go was easier to understand. As also expected, Extended VR Sliding was faster and more accurate for targets that were in contact with the scene.

Go-Go+PRISM and Ray-Casting-with-Reeling did not perform as well as Go-Go in terms of positioning time and error measure. The adaptive speed control in Go-Go+PRISM did not improve the accuracy of the positioning tasks. Eight users found Go-Go+PRISM hard to use. They claimed that they did not like the offset between the virtual controller and object positions. In the Ray-Casting-with-Reeling technique, I used a constant speed to change the length of controller ray. This could significantly slow down the task when the target is far away from the object in terms of (visual) depth. While a non-linear speed control technique could be used to change the length of the ray, this would introduce the issue of overshooting behaviours. Therefore, I did not include these two techniques in user study 2.

Based on the results of user study 1 and users' feedback, I modified my Extended VR Sliding technique, where I kept the 2DOF sliding and 2DOF lifting phases.

Yet, when the users switched to the final adjustment phase, they could move the object in 3 DOFs simultaneously with free hand movement. Still, users had the freedom to use each phase of Extended VR Sliding for a shorter or longer period of time, as they saw fit and as the task demanded. In user study 2, the ease of interaction of Extended VR Sliding received three more response (10) above neutral than user study 1 (7). Moreover, three more participants rated the fatigue level of Extended VR Sliding neutral or low (11 in user study 2) than user study 1 (8). According to the feedback, this mode change was easy to understand and use and thus worked better than in the first version.

The results of user study 2 showed that the simplified Extended VR Sliding technique significantly improved the accuracy of placement for floating targets. Error measure for such condition dropped from 0.671 in user study 1 to 0.310 in user study 2. If the target was floating, users could quickly slide the object to a surface near the target, lift the object close to the target and then fine adjust with 3D hand movements. Although the Extended VR Sliding, Floating combination was still slower than with Go-Go, I believe that users could improve their speed with more training. For targets in contact, simplified Extended VR Sliding still performed best for time and accuracy. The Extended VR Sliding, Contact combination was faster and more accurate than any combinations of Go-Go.

Typically, the Extended VR Sliding technique requires a surface in the scene to slide the object on. If there is no object in the scene, the users can still slide the object parallel to their view plane. In the lifting phase, the user then “lifts” the object in the view direction, i.e., moves the object closer or further away from them. This makes Extended VR Sliding more general.

Extended VR Sliding’s performance was robust to depth distance changes. The Small and Large condition did not have any significant effect on either Extended VR Sliding’s positioning time or error measure in any of the two studies. However, as I increased the depth distance change in user study 2, the impact of depth distance change on Go-Go started to appear. In user study 2, Go-Go, Small was the fastest yet Go-Go, Large was the least accurate. This exposes a major limitation of Go-Go, as the maximum depth distance change that it supports is limited. Especially when users are sitting down, the virtual controller can only be extended by a limited distance. This limit is even smaller for users with shorter arms. If a scene contains an object that is too far

away, users then cannot reach it even with full arm extension. The only option would be changing the constant used in the Go-Go technique. Yet, this might well affect the accuracy of the technique negatively. Also, full arm extension could generate more fatigue.

In the first study, the smallest error measure was 0.15, which occurred in the Extending Sliding, Contact condition. In the second study, the smallest error measure was 0.20 in the same condition. The increase of error in the second study was likely due to the increase of object distance in depth. The error measure was a relative measure, calculated from the error distance over the object size. The object size was 30 cm in the second study, which means the error distance was 6 cm on average. When positioning a 30 cm large object to a target position 5 m away in depth, the object appears to be fairly small. A 6 cm error distance is reasonable, even for targets in contact.

When analyzing the performance of the 5 trials in each condition, I did not find an obvious pattern. The performance did not improve over the trials. This is likely because I did not use repetitions for the trials. If I had used repetitions, it would be interesting to identify the increase in performance over trials. Although fatigue could have an effect, I would assume that participants would improve their performance over the five trials. Therefore, I would have a better idea of the learning curve of each condition.

I used 12 participants for both studies. I acknowledge that for the first study, which had a 4x2x2 design, I did not have enough participants for a counterbalanced design. Even with balanced Latin square, at least 16 participants should have been used. For the second study, a perfectly counterbalanced design would require the participant number to be a multiple of 8. Therefore, there could be some order effects that may have affected the results. If one condition is more often performed after another condition, fatigue could have a noticeable effect on performance, and therefore bias the results. However, judging from the significance of the data, those effects should be small.

In both studies, I used the trigger button on the user's non-dominant hand for confirmation. This effectively reduced the Heisenberg effect [Bowman et al., 2001], or as the users described in their freeform feedback, "the hand shaking effect" caused by releasing a button. This matched the results of [Batmaz et al., 2019]. Unlike the study in

Chapter 4 where participants used their second hand for stabilization, participants in both studies of this chapter did not do the same. Results in user study 1 showed that Go-Go was faster and more accurate than Go-Go+PRISM. This established that when the non-dominant hand was used for confirmation, slowing down the object movement could be redundant. Therefore, I kept using the non-dominant hand for confirmation in the simplified Extended VR Sliding technique. Participants were able to learn using the non-dominant hand for confirmation without any issues.

Teather et al. proposed some guidelines for 3D positioning [Teather et al., 2007]. They proposed to avoid 3D handles or widgets for 3D positioning. In desktop environments, novice users can manipulate 3D objects more effectively without handles. Sliding is an obvious example [Oh et al., 2005]. I think this guideline still holds for VR. Although using 3D widgets in VR could improve the positioning accuracy for closer objects [Mendes et al., 2017], the excessive selection of 3D widgets can slow down positioning especially for distant objects. Therefore, I did not include the 3D widgets condition in my experiment.

Poupyrev et al. found that for object positioning without a substantial depth distance change, ray-casting (without reeling) performs significantly faster than Go-Go [Poupyrev et al., 1998]. My first user study did not yield the same result, as even the Small condition of depth distance change required the users to use ray-casting with reeling. The different implementation of Go-Go from [Poupyrev et al., 1996] also could have contributed to the different result, as it decreased the lateral hand movement. I made Extended VR Sliding technique robust for depth distance change, which allows object positioning at various distance.

Identifying an appropriate mapping for mid-air VR manipulation that achieves satisfactory precision is still an open problem [Mendes et al., 2019]. I proposed a mapping that significantly improves the accuracy of 3D positioning for targets in contact. I used constraints, which are not widely used in VR, to slide objects on surfaces and lift objects off surfaces. Different DOFs were manipulated separately. Both object sliding and lifting are 2DOF manipulation. Extended VR Sliding even works for scenes without surfaces. For fine adjustment, I allowed 3DOF free hand movement, which was easy to learn and preferred by users.

In conclusion, I designed and implemented an Extended VR Sliding technique for 3D positioning. The results of two user studies showed that Extended VR Sliding significantly improved the accuracy for 3D positioning tasks for targets in contact with the scene. For targets that were in contact with the scene, Extended VR Sliding was also significantly faster than any other technique. Thus, my new technique could improve the accuracy of 3D positioning tasks in applications such as 3D modelling in VR.

Chapter 6.

Conclusion and Future Work

3D object manipulation is one of the fundamental 3DUI tasks for interaction in virtual environments The goal of my thesis is to create better user interfaces for 3D object manipulation, with a focus on 3D positioning. In this thesis, I focussed on precise 3D positioning task that require moving a selected object to a target position. I investigated various aspects of 3D positioning. I explore 3D positioning tasks both on in desktop and in VR systems.

There has been substantial research in the field of 3D manipulation. After doing literature review, I identified some gaps in the current industry standard 3D software and literature of 3D manipulation. On the desktop system, the majority of 3D modelling software use 3D widgets [Conner et al., 1992] for object positioning. The efficiency of 3D widgets could be improved. Contact-plane sliding [Oh et al., 2005] outperformed 3D widgets, yet had some limitations on objects' positions. Also, with 3D widgets or even contact-plane sliding, users can only select and position a partially visible object unless camera navigation or multiple viewpoints are enabled. Despite that the mouse is a great input device for precise 3D manipulation [Bérard et al., 2009], the most commonly used input devices for object manipulation in VR are 3DOF controllers. The use of mouse in immersive VR is rarely studied. Moreover, when using a 3DOF controller as input device, the most appropriate mappings have yet to be found for fast and precise mid-air object positioning [Mendes et al., 2019].

3D positioning is widely used in applications such as 3D games, interior and exterior design, industrial modeling, 3D animations, and homebuilder applications. In many of these applications, speed and accuracy are essential. My goal is to have 3D content designers, game designers, or even novice users benefit from the improved ease of use, speed, and accuracy of 3D positioning tasks. Thus, I used measured time and error and used questionnaires in all the studies. My work will also provide guidelines for designing and developing 3D modelling software for both desktop and VR systems.

In Chapter 2, I presented two novel 3D positioning techniques that are fast and easy to use. I extended contact-plane sliding [Oh et al., 2005] with the new SHIFT-Sliding and DEPTH-POP methods. The results of the user studies showed that for novice users the new methods are faster for 3D positioning compared to the standard widget-based approach [Conner et al., 1992]. Both new methods profoundly enhanced the ease and efficiency of 3D manipulation with 2D input devices. This answers **RQ1: "How much does extended sliding improve the speed/accuracy of 3D positioning in a desktop system?"** in a positive way, as the extended sliding techniques did improve 3D positioning in a desktop system.

The current implementation of DEPTH-POP could be slow in scenes with high depth complexity on lower-end graphics hardware. In scenes with many hidden layers, the large number of small fragments could lead to a large amount of potential solutions. Many of those solutions might be meaningless for interaction. In the future, I intend to optimize the algorithm to deal better with such cases through appropriate pruning. Towards this, I intend to add the SHIFT-Sliding and DEPTH-POP techniques to modern 3D CAD software.

In Chapter 3, I presented the new Control-Depth selection method, which enabled users to select hidden objects in arbitrary scenes without resorting to (potentially time-consuming) camera navigation, by iteratively "peeling away" layers [Everitt, 2001] in front of a target object and then simply selecting it. I also proposed a new extension to the contact-plane sliding technique, which uses a transparency mask to facilitate object sliding, so that the user can see the object during positioning, even if the object slides behind other ones.

I performed a user study to compare the performance of my new selection and transparency sliding techniques with common 3D manipulation widgets. Users had to select hidden objects and move and precisely position them at hidden target positions. The results showed that even though it took an extra step to complete the task in the contact-plane sliding condition (selection and positioning), the combined time was still significantly less than with common 3D widgets. Users found my techniques easy to learn and use. This addresses **RQ2: "How much does layer-based transparency improve the speed/accuracy of selection and positioning of hidden objects?"**

positively, as users were able to select and slide hidden object easily with the new transparency techniques.

In the future, I plan to investigate methods to optimize the implementation of Control-Depth selection and reduce the impact of limitations of the z-buffer, specifically z-fighting. I may also investigate how to improve the rendering of semi-transparent effects for different textures and geometries. Additionally, I will check if more complex scene geometry affects the users' perception of the scene while using my techniques. In order to further evaluate the efficiency and simplicity of the techniques, I would also like to perform another experiment with both novices and participants that are familiar with desktop-based 3D interaction. There, I would evaluate how much benefits my techniques have for experts and subsequently include the transparency techniques into 3D CAD software.

In Chapter 4, I compared three input and two cursor display methods for precise positioning in the HTC Vive. As the mouse performed in general better than both the controller and the trackpad, this answers **RQ3: "How do input devices compare for object sliding in VR?"** As the 2DOF nature of the technique matches 2DOF input devices such as the mouse, this result might be partially due to the use of a sliding technique I choose for the evaluation. Yet, I think that the result confirms that the mouse is a good input device for precise 3D positioning in an HMD-based VR system in situations where users have a stable surface for the mouse available, such as a table or a chair-integrated mouse pad.

Regarding **RQ4: "Which cursor display performs better for object sliding in VR?",** cursor display did not have a significant effect on either completion time or error measure. Yet, users were more comfortable using the stereo cursor for 3D positioning in the HMD. When designing an interface for HMD, both my and Teather et al.'s results [Teather et al., 2013] have to be taken into consideration. I assume that stereo cursor would outperform one-eyed cursor in a combined task requiring both selection and positioning, but this needs to be verified by a future study.

Some may argue that the mouse is not suitable for immersive virtual reality. However, I think it is currently not practical to expect people to use an HMD in a standing pose for an extended period of time, say for a full workday. Therefore, there is a need for

user interfaces for users that use an HMD in a seated posture, or at least in front of a standing desk. The choice of input device depends on the tasks and platforms. With a sliding technique, the mouse is the better device. This provides guidance for the design of 3D modelling software in VR. In other words, 3D CAD applications should setup the VR system so that they can be used sitting down with a mouse, if possible.

In Chapter 5, I implemented SHIFT-Sliding in an HTC Vive-based VR system to enable full 3D positioning in more general scenes. I identified appropriate mappings for Extended VR Sliding with the 3DOF controller as input device and evaluated positioning in a comparative study involving full 3DOF techniques. The results of two user studies showed that Extended VR Sliding significantly improved the accuracy for 3D positioning tasks for targets in contact with the scene. For targets that were in contact with the scene, Extended VR Sliding was also significantly faster than any other technique. Since Extended VR Sliding did not require full arm extension, it also caused less fatigue. All these outcomes address **RQ5: "How much does extended sliding improve the speed/accuracy of 3D positioning in VR?"**

In the future, I plan to further improve the controller mappings of Extended VR Sliding. I would like to make the technique even easier to understand and use, especially for floating objects. I intend to include SHIFT-Sliding implementations in 3D modelling software in VR. I would also like to explore 3D positioning techniques that allow users to move around freely, e.g., let users move around and move furniture in a VR home builder application. With precise reconstruction of the real world and proper tracking of the input device, sliding can even be applied to augmented reality applications, allowing, e.g., users to view the results of homebuilding/scene modifications accurately and directly in the real world.

Here I list the major contributions of my thesis.

1. Presented two easy-to-use constraint-based 3D positioning techniques for desktop systems that extended object sliding and outperformed 3D widgets in speed, accuracy, and usability.

2. Presented two easy-to-use transparency-based techniques for desktop systems that enabled selecting and positioning of fully hidden objects without camera navigation and outperformed 3D widgets in speed and usability.

3. Provided guidelines for 3D modelling software in VR with headsets: 3D applications should use stereo cursor display to generate less fatigue and use the mouse as input device if the applications can be used in a seated position. If possible, a stable surface should be provided to users to rest their hands.

4. Proposed new mappings for mid-air 3D object positioning in VR with controllers that improved accuracy. Provided guidelines for creating fast and accurate 3D applications that use 3DOF controllers as input devices: use constraints if possible; avoid 3D widgets for manipulation of distant objects; use non-dominant hand for confirmation; avoid controller-object offset; allow 3DOF free hand fine adjustment.

In my thesis, I filled in the gaps I found in the literature. I explored different combinations of platform, input device, and technique for 3D manipulation tasks, with a focus on precise 3D object positioning. Besides performance measures such as speed and precision, I also prioritized the usability of the 3D positioning techniques. I found some combinations of input device and technique that are easy to learn and use and generate less fatigue, while still being fast and accurate.

I presented several novel techniques for 3D positioning. The results of my work should be will beneficial to both novice and expert users. Novice users have less experience in 3D modelling, and some have weaker spatial abilities. To address the needs of this user population, I made my techniques natural and easy for users to understand and learn. On the other hand, expert users use 3D modelling software more frequently. They may learn techniques quicker, yet my novel techniques will still provide benefits as they improve their efficiency and accuracy in their modelling work. Therefore, I intend to implement my techniques in existing 3D CAD and VR software systems.

Besides proposing new techniques, I also provided some guidelines for designing and developing 3D CAD software, for both desktop and VR systems. In the future, I plan to resolve some of the technical challenges associated with the techniques. I will also further improve the usability of the techniques, making them easy to learn and use. Overall, I hope that my work will improve the usability, efficiency, and accuracy of object positioning in 3D user interfaces.

References

Agustina and C. Sun, 2013, February. Xpointer: an X-ray telepointer for relaxed-space-time Wysiwis and unconstrained collaborative 3D design systems. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work* (pp. 729-740). ACM.

Argelaguet, F. and Andujar, C., 2009, November. Visual feedback techniques for virtual pointing on stereoscopic displays. In *Proceedings of the 16th ACM Symposium on Virtual Reality Software and Technology* (pp. 163-170). ACM.

Argelaguet, F. and Andujar, C., 2013. A survey of 3D object selection techniques for virtual environments. *Computers & Graphics*, *37*(3), pp.121-136.

Au, O.K.C., Tai, C.L. and Fu, H., 2012, May. Multitouch gestures for constrained transformation of 3d objects. In *Computer* Graphics *Forum* (Vol. 31, No. 2pt3, pp. 651-660). Oxford, UK: Blackwell Publishing Ltd.

Auteri, C., Guerra, M. and Frees, S., 2013. Increasing precision for extended reach 3D manipulation. *Int J Virtual Real*, *12*(1), pp.66-73.

Ayatsuka, Y., Matsuoka, S. and Rekimoto, J., 1996, November. Penumbrae for 3 D interactions. In *ACM Symposium on User Interface Software and Technology* (pp. 165-166).

Bade, R., Ritter, F. and Preim, B., 2005, August. Usability comparison of mouse-based interaction techniques for predictable 3d rotation. In *International Symposium on Smart Graphics* (pp. 138-150). Springer, Berlin, Heidelberg.

Batmaz, A.U., and Stuerzlinger, W. Effects of 3D Rotational Jitter and Selection Methods on 3D Pointing Tasks, *Workshop on Novel Input Devices and Interaction Techniques (NIDIT) at IEEE VR 2019*, 4 pages, March 2019.

Bavoil, L. and Myers, K., 2008. Order independent transparency with dual depth peeling. *NVIDIA OpenGL SDK*, pp.1-12.

Bérard, F., Ip, J., Benovoy, M., El-Shimy, D., Blum, J.R. and Cooperstock, J.R., 2009, August. Did “Minority Report” get it wrong? Superiority of the mouse over 3D input devices in a 3D placement task. In *IFIP Conference on Human-Computer Interaction* (pp. 400-414). Springer, Berlin, Heidelberg.

Besançon, L., Issartel, P., Ammi, M. and Isenberg, T., 2017, May. Mouse, tactile, and tangible input for 3D manipulation. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (pp. 4727-4740).

Bier, E.A., 1990, February. Snap-dragging in three dimensions. In *ACM SIGGRAPH Computer Graphics* (Vol. 24, No. 2, pp. 193-204). ACM.

Bier, E.A., Stone, M.C., Pier, K., Buxton, W. and DeRose, T.D., 1993, September. Toolglass and magic lenses: the see-through interface. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques* (pp. 73-80).

Bowman, D.A. and Hodges, L.F., 1997, April. An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environments. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics* (pp. 35-38). ACM.

Bowman, D., Wingrave, C., Campbell, J. and Ly, V., 2001. Using pinch gloves (tm) for both natural and abstract interaction techniques in virtual environments. *HCI International*, pages 629-633, 2001.

Bowman, D., Kruijff, E., LaViola Jr, J.J. and Poupyrev, I.P., 2004. *3D User interfaces: theory and practice, CourseSmart eTextbook*. Addison-Wesley.

Brown, M.A., Stuerzlinger, W. and Filho, E.M., 2014. The performance of un-instrumented in-air pointing. In *Proceedings of Graphics Interface 2014* (pp. 59-66).

Bukowski, R.W. and Séquin, C.H., 1995, April. Object associations: a simple and practical approach to virtual 3D manipulation. In *Proceedings of the 1995 Symposium on Interactive 3D Graphics* (pp. 131-ff). ACM.

Butterworth, J., Davidson, A., Hench, S., and Olano, M.T., 1992, June. 3DM: A three dimensional modeler using a head-mounted display. In *Proceedings of the 1992 Symposium on Interactive 3D Graphics* (pp. 135-138). ACM.

Caputo, F.M., Emporio, M. and Giachetti, A., 2018. The Smart Pin: An effective tool for object manipulation in immersive virtual reality environments. *Computers & Graphics*, *74*, pp.225-233.

Carifio, J. and Perla, R., 2008. Resolving the 50-year debate around using and misusing Likert scales. *Medical Education*, *42*(12), pp.1150-1152.

Chittaro, L. and Scagnetto, I., 2001, November. Is semitransparency useful for navigating virtual environments?. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology* (pp. 159-166).

Cho, I. and Wartell, Z., 2015, March. Evaluation of a bimanual simultaneous 7dof interaction technique in virtual environments. In *3D User Interfaces (3DUI), 2015 IEEE Symposium on* (pp. 133-136). IEEE.

Coffin, C. and Hollerer, T., 2006, March. Interactive perspective cut-away views for general 3d scenes. In *3D User Interfaces (3DUI'06)* (pp. 25-28). IEEE.

Conner, B.D., Snibbe, S.S., Herndon, K.P., Robbins, D.C., Zeleznik, R.C. and Van Dam, A., 1992, June. Three-dimensional widgets. In *Proceedings of the 1992 Symposium on Interactive 3D Graphics* (pp. 183-188). ACM.

Davidson, P.L. and Han, J.Y., 2008, October. Extending 2D object arrangement with pressure-sensitive layering cues. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology* (pp. 87-90).

Diepstraten, J., Weiskopf, D. and Ertl, T., 2002, September. Transparency in interactive technical illustrations. In *Computer Graphics Forum* (Vol. 21, No. 3, pp. 317-325). Oxford, UK: Blackwell Publishing, Inc.

Elmqvist, N., Assarsson, U. and Tsigas, P., 2007, September. Employing dynamic transparency for 3D occlusion management: Design issues and evaluation. In *IFIP Conference on Human-Computer Interaction* (pp. 532-545). Springer, Berlin, Heidelberg.

Elmqvist, N. and Tsigas, P., 2008. A taxonomy of 3d occlusion management for visualization. *IEEE Transactions on Visualization and Computer Graphics*, *14*(5), pp.1095-1109.

Englund, R. and Ropinski, T., 2016, November. Evaluating the perception of semi-transparent structures in direct volume rendering techniques. In *SIGGRAPH ASIA 2016 Symposium on Visualization* (pp. 1-8).

Everitt, C., 2001. Interactive order-independent transparency. *White paper, nVIDIA*, *2*(6), p.7.

Fernquist, J., Shoemaker, G. and Booth, K.S., 2011, September. "Oh Snap"–Helping Users Align Digital Objects on Touch Interfaces. In *IFIP Conference on Human-Computer Interaction* (pp. 338-355). Springer, Berlin, Heidelberg.

Frees, S. and Kessler, G.D., 2005, March. Precise and rapid interaction through scaled manipulation in immersive virtual environments. In *Virtual Reality, 2005. Proceedings. VR 2005. IEEE* (pp. 99-106). IEEE.

Frees, S., Kessler, G.D. and Kay, E., 2007. PRISM interaction for enhancing control in immersive virtual environments. *ACM Transactions on Computer-Human Interaction (TOCHI)*, *14*(1), p.2.

Froehlich, B., Hochstrate, J., Skuk, V. and Huckauf, A., 2006, April. The globefish and the globemouse: two new six degree of freedom input devices for graphics applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 191-199).

Fu, C.W., Xia, J. and He, Y., 2010, April. LayerPaint: a multi-layer interactive 3D painting interface. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 811-820).

George, D., 2011. *SPSS for windows step by step: A simple study guide and reference, 17.0 update, 10/e*. Pearson Education India.

Glueck, M., Crane, K., Anderson, S., Rutnik, A. and Khan, A., 2009, February. Multiscale 3D reference visualization. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games* (pp. 225-232).

Gutwin, C., Dyck, J. and Fedak, C., 2003. The Effects of Dynamic Transparency on Targeting Performance. In *Graphics Interface* (pp. 105-112).

Hachet, M., Guitton, P. and Reuter, P., 2003, October. The CAT for efficient 2D and 3D interaction as an alternative to mouse adaptations. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology* (pp. 225-112).

Hancock, M., Carpendale, S. and Cockburn, A., 2007, April. Shallow-depth 3d interaction: design and evaluation of one-, two-and three-touch techniques. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 1147-1156).

Harrison, B.L., Ishii, H., Vicente, K.J. and Buxton, W., 1995, May. Transparent layered user interfaces: An evaluation of a display design to enhance focused and divided attention. In *CHI* (Vol. 95, pp. 317-324).

Harrison, B.L., Kurtenbach, G. and Vicente, K.J., 1995, December. An experimental evaluation of transparent user interface tools and information content. In *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology* (pp. 81-90).

Heer, J. and Bostock, M., 2010, April. Crowdsourcing graphical perception: using mechanical turk to assess visualization design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 203-212).

Herndon, K.P., Zeleznik, R.C., Robbins, D.C., Conner, D.B., Snibbe, S.S. and Van Dam, A., 1992, December. Interactive shadows. In *Proceedings of the 5th Annual ACM Symposium on User Interface Software and Technology* (pp. 1-6).

Herndon, K.P., van Dam, A. and Gleicher, M., 1994. The challenges of 3D interaction: a CHI'94 workshop. *ACM SIGCHI Bulletin*, *26*(4), pp.36-43.

Herrlich, M., Walther-Franks, B. and Malaka, R., 2011, July. Integrated rotation and translation for 3D manipulation on multi-touch interactive surfaces. In *International Symposium on Smart Graphics* (pp. 146-154). Springer, Berlin, Heidelberg.

Hill, A. and Johnson, A., 2008, March. Withindows: A framework for transitional desktop and immersive user interfaces. In *2008 IEEE Symposium on 3D User Interfaces*(pp. 3-10). IEEE.

Hincapié-Ramos, J.D., Guo, X., Moghadasian, P. and Irani, P., 2014, April. Consumed endurance: a metric to quantify arm fatigue of mid-air interactions. In *Proceedings of the 32nd Annual ACM Conference on Human Factors in Computing Systems* (pp. 1063-1072). ACM.

Hinckley, K., Pausch, R., Goble, J.C. and Kassell, N.F., 1994, November. A survey of design issues in spatial input. In *Proceedings of the 7th Annual ACM Symposium on User Interface Software and Technology* (pp. 213-222).

Hinckley, K., Tullio, J., Pausch, R., Proffitt, D. and Kassell, N., 1997, October. Usability analysis of 3D rotation techniques. In *Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology* (pp. 1-10). ACM.

Hoppe, A.H., van de Camp, F. and Stiefelhagen, R., 2017, July. Interaction with three dimensional objects on diverse input and output devices: A survey. In *International Conference on Human-Computer Interaction* (pp. 130-139). Springer, Cham.

Igarashi, T. and Mitani, J., 2010. Apparent layer operations for the manipulation of deformable objects. In *ACM SIGGRAPH 2010 papers* (pp. 1-7).

Ishak, E.W. and Feiner, S.K., 2004, October. Interacting with hidden content using content-aware free-space transparency. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology* (pp. 189-192). ACM.

Jacob, R.J., Sibert, L.E., McFarlane, D.C. and Mullen Jr, M.P., 1994. Integrality and separability of input devices. *ACM Transactions on Computer-Human Interaction (TOCHI)*, *1*(1), pp.3-26.

Jang, S., Stuerzlinger, W., Ambike, S. and Ramani, K., 2017, May. Modeling cumulative arm fatigue in mid-air interaction based on perceived exertion and kinetics of arm motion. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (pp. 3328-3339). ACM.

Jáuregui, D.A.G., Argelaguet, F. and Lecuyer, A., 2012, March. Design and evaluation of 3D cursors and motion parallax for the exploration of desktop virtual environments. In *2012 IEEE Symposium on 3D User Interfaces (3DUI)* (pp. 69-76). IEEE.

Javed, W., Kim, K., Ghani, S. and Elmqvist, N., 2011, September. Evaluating physical/virtual occlusion management techniques for horizontal displays. In *IFIP Conference on Human-Computer Interaction* (pp. 391-408). Springer, Berlin, Heidelberg.

Kitamura, Y., Ogata, S. and Kishino, F., 2002, November. A manipulation environment of virtual and real objects using a magnetic metaphor. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology* (pp. 201-207). ACM.

Kopper, R., Silva, M.G., McMahan, R.P. and Bowman, D.A., 2008. Increasing the precision of distant pointing for large high-resolution displays. Department of Computer Science, Virginia Polytechnic Institute & State University.

Krichenbauer, M., Yamamoto, G., Taketom, T., Sandor, C. and Kato, H., 2017. Augmented reality versus virtual reality for 3d object manipulation. *IEEE Transactions on Visualization and Computer Graphics*, *24*(2), pp.1038-1048.

Kruger, R., Carpendale, S., Scott, S.D. and Tang, A., 2005, April. Fluid integration of rotation and translation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 601-610).

Liang, J. and Green, M., 1993, August. Geometric modeling using six degrees of freedom input devices. In *3rd Int'l Conference on CAD and Computer Graphics* (pp. 217-222).

Luboschik, M., Radloff, A. and Schumann, H., 2010, May. A new weaving technique for handling overlapping regions. In *Proceedings of the International Conference on Advanced Visual Interfaces* (pp. 25-32).

Machuca, M.D.B., Sun, J., Pham, D.M. and Stuerzlinger, W., 2018, March. Fluid VR: Extended Object Associations for Automatic Mode Switching in Virtual Reality. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)* (pp. 846-847). IEEE.

MacKenzie, I.S., 1992. Fitts' law as a research and design tool in human-computer interaction. *Human-Computer Interaction*, *7*(1), pp.91-139.

MacKenzie, I.S., 2012. *Human-computer interaction: An empirical research perspective*. Newnes.

Martinet, A., Casiez, G. and Grisoni, L., 2010, March. The design and evaluation of 3d positioning techniques for multi-touch displays. In *2010 IEEE Symposium on 3D User Interfaces (3DUI)* (pp. 115-118). IEEE.

Martinet, A., Casiez, G. and Grisoni, L., 2010, November. The effect of DOF separation in 3D manipulation tasks with multi-touch displays. In *Proceedings of the 17th ACM Symposium on Virtual Reality Software and Technology* (pp. 111-118). ACM.

Masliah, M.R. and Milgram, P., 2000, April. Measuring the allocation of control in a 6 degree-of-freedom docking experiment. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 25-32). ACM.

McMahan, R.P., Gorton, D., Gresock, J., McConnell, W. and Bowman, D.A., 2006, November. Separating the effects of level of immersion and 3D interaction techniques. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology* (pp. 108-111).

Mendes, D., Sousa, M., Lorena, R., Ferreira, A. and Jorge, J., 2017, November. Using custom transformation axes for mid-air manipulation of 3D virtual objects. In *Proceedings of the 23rd ACM Symposium on Virtual Reality Software and Technology* (p. 27). ACM.

Mendes, D., Caputo, F.M., Giachetti, A., Ferreira, A. and Jorge, J., 2019, February. A survey on 3D virtual object manipulation: From the desktop to immersive virtual environments. In *Computer Graphics Forum* (Vol. 38, No. 1, pp. 21-45).

Mine, M.R., Brooks Jr, F.P. and Sequin, C.H., 1997, August. Moving objects in space: exploiting proprioception in virtual-environment interaction. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (pp. 19-26).

Norman, G., 2010. Likert scales, levels of measurement and the “laws” of statistics. *Advances in Health Sciences Education*, *15*(5), pp.625-632.

Oh, J.Y. and Stuerzlinger, W., 2005, May. Moving objects with 2D input devices in CAD systems and desktop virtual environments. In *Proceedings of Graphics Interface 2005* (pp. 195-202). Canadian Human-Computer Communications Society.

Ortega, M. and Vincent, T., 2014, April. Direct drawing on 3D shapes with automated camera control. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 2047-2050).

Pierce, J.S., Forsberg, A.S., Conway, M.J., Hong, S., Zeleznik, R.C. and Mine, M.R., 1997, April. Image plane interaction techniques in 3D immersive environments. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics* (pp. 39-43). ACM.

Poupyrev, I., Billinghurst, M., Weghorst, S. and Ichikawa, T., 1996, November. The go-go interaction technique: non-linear mapping for direct manipulation in VR. In *ACM Symposium on User Interface Software and Technology* (pp. 79-80).

Poupyrev, I., Ichikawa, T., Weghorst, S. and Billinghurst, M., 1998, August. Egocentric object manipulation in virtual environments: empirical evaluation of interaction techniques. In *Computer Graphics Forum* (Vol. 17, No. 3, pp. 41-52). Oxford, UK and Boston, USA: Blackwell Publishers Ltd.

Ramos, G., Robertson, G., Czerwinski, M., Tan, D., Baudisch, P., Hinckley, K. and Agrawala, M., 2006, May. Tumble! Splat! helping users access and manipulate occluded content in 2D drawings. In *Proceedings of the Working Conference on Advanced Visual Interfaces* (pp. 428-435).

Reisman, J.L., Davidson, P.L. and Han, J.Y., 2009, October. A screen-space formulation for 2D and 3D direct manipulation. In *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology* (pp. 69-78).

Scheurich, D. and Stuerzlinger, W., 2013, September. A one-handed multi-touch method for 3D rotations. In *IFIP Conference on Human-Computer Interaction* (pp. 56-69). Springer, Berlin, Heidelberg.

Schmidt, R., Singh, K. and Balakrishnan, R., 2008, April. Sketching and composing widgets for 3d manipulation. In *Computer graphics forum* (Vol. 27, No. 2, pp. 301-310). Oxford, UK: Blackwell Publishing Ltd.

Shaw, C.D. and Green, M., 1997. THRED: a two-handed design system. *Multimedia Systems*, *5*(2), pp.126-139.

Shoemake, K., 1992, September. ARCBALL: a user interface for specifying three-dimensional orientation using a mouse. In *Graphics Interface* (Vol. 92, pp. 151-156).

Shuralyov, D. and Stuerzlinger, W., 2011, March. A 3D desktop puzzle assembly system. In *2011 IEEE Symposium on 3D User Interfaces (3DUI)* (pp. 139-140). IEEE.

Smith, G., Stuerzlinger, W., Salzman, T., Watson, B. and Buchanan, J., 2001, June. 3D scene manipulation with 2D devices and constraints. In *Graphics Interface* (Vol. 1, pp. 135-142).

Song, P., Goh, W.B., Hutama, W., Fu, C.W. and Liu, X., 2012, May. A handle bar metaphor for virtual object manipulation with mid-air interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 1297-1306). ACM.

Stoakley, R., Conway, M.J., and Pausch, R., (1995). Virtual reality on a WIM: interactive worlds in miniature. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 265–272). ACM Press/Addison-Wesley Publishing Co.

Strauss, P.S. and Carey, R., 1992. An object-oriented 3D graphics toolkit. *ACM SIGGRAPH Computer Graphics*, *26*(2), pp.341-349.

Strothoff, S., Valkov, D. and Hinrichs, K., 2011, November. Triangle cursor: Interactions with objects above the tabletop. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces* (pp. 111-119). ACM.

Stuerzlinger, W. and Wingrave, C.A., 2011. The value of constraints for 3D user interfaces. In *Virtual Realities* (pp. 203-223). Springer, Vienna.

Sun, J., Stuerzlinger, W. and Shuralyov, D., 2016, October. Shift-sliding and depth-pop for 3D positioning. In *Proceedings of the 2016 Symposium on Spatial User Interaction* (pp. 69-78). ACM.

Sun, J., Stuerzlinger, W. and Riecke, B.E., 2018, August. Comparing input methods and cursors for 3D positioning with head-mounted displays. In *Proceedings of the 15th ACM Symposium on Applied Perception* (p. 8). ACM.

Sun, J. and Stuerzlinger, W., 2019, June. Selecting and Sliding Hidden Objects in 3D Desktop Environments. In *Proceedings of the 45th Graphics Interface Conference on Proceedings of Graphics Interface 2019* (pp. 1-8). Canadian Human-Computer Communications Society.

Sun, J. and Stuerzlinger, W., 2019. Extended Sliding in Virtual Reality, In *Proceedings of VRST '19: 25th ACM Symposium on Virtual Reality Software and Technology* (VRST '19). ACM.

Teather, R.J. and Stuerzlinger, W., 2007, November. Guidelines for 3D positioning techniques. In *Proceedings of the 2007 Conference on Future Play* (pp. 61-68). ACM.

Teather, R.J. and Stuerzlinger, W., 2013, April. Pointing at 3d target projections with one-eyed and stereo cursors. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 159-168). ACM.

Van Emmerik, M.J., 1990, December. A direct manipulation technique for specifying 3D object transformations with a 2D input device. In *Computer Graphics Forum* (Vol. 9, No. 4, pp. 355-361). Oxford, UK: Blackwell Publishing Ltd.

Vanacken, L., Grossman, T. and Coninx, K., 2007, March. Exploring the effects of environment density and target visibility on object selection in 3D virtual environments. In *2007 IEEE Symposium on 3D User Interfaces*. IEEE.

Venolia, D., 1993, May. Facile 3D direct manipulation. In *Proceedings of the INTERACT'93 and CHI'93* Conference *on Human Factors in Computing Systems* (pp. 31-36). ACM.

Vishton, P.M. and Cutting, J.E., 1995. Wayfinding, displacements, and mental maps: velocity fields are not typically used to determine one's aimpoint. *Journal of Experimental Psychology: Human Perception and Performance*, *21*(5), p.978.

Vogel, D. and Balakrishnan, R., 2005, October. Distant freehand pointing and clicking on very large, high resolution displays. In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology* (pp. 33-42). ACM.

Vuibert, V., Stuerzlinger, W. and Cooperstock, J.R., 2015, August. Evaluation of docking task performance using mid-air interaction techniques. In *Proceedings of the 3rd ACM Symposium on Spatial User Interaction* (pp. 44-52).

Ware, C. and Jessome, D.R., 1988. Using the bat: A six-dimensional mouse for object placement. *IEEE Computer Graphics and Applications*, *8*(6), pp.65-70.

Ware, C. and Lowther, K., 1997. Selection using a one-eyed cursor in a fish tank VR environment. *ACM Transactions on Computer-Human Interaction (TOCHI)*, *4*(4), pp.309-322.

Webb, A.M., Kerne, A., Brown, Z., Kim, J.H. and Kellogg, E., 2016, November. Layerfish: Bimanual layering with a fisheye in-place. In *Proceedings of the 2016 ACM International Conference on Interactive Surfaces and Spaces* (pp. 189-198).

Whyte, J., Bouchlaghem, N., Thorpe, A. and McCaffer, R., 2000. From CAD to virtual reality: modelling approaches, data exchange and interactive 3D building design tools. *Automation in Construction*, *10*(1), pp.43-55.

Wickens, C. and Hollands, J., 1999. Spatial displays. *Engineering Psychology and Human Performance, Prentice-Hall*, *3*.

Wilkes, C. and Bowman, D.A., 2008, October. Advantages of velocity-based scaling for distant 3D manipulation. In *Proceedings of the 2008 ACM Symposium on Virtual Reality Software and Technology* (pp. 23-29). ACM.

Wobbrock, J.O., Findlater, L., Gergle, D. and Higgins, J.J., 2011, May. The aligned rank transform for nonparametric factorial analyses using only anova procedures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 143-146).

Wunsch, P. and Hirzinger, G., 1997, April. Real-time visual tracking of 3D objects with dynamic handling of occlusion. In *Proceedings of International Conference on Robotics and Automation* (Vol. 4, pp. 2868-2873). IEEE.

Zhai, S., Buxton, W. and Milgram, P., 1994, April. The “Silk Cursor” investigating transparency for 3D target acquisition. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 459-464).

Zhai, S., Buxton, W. and Milgram, P., 1996. The partial-occlusion effect: Utilizing semitransparency in 3D human-computer interaction. *ACM Transactions on Computer-Human Interaction (TOCHI)*, *3*(3), pp.254-284.

Zhai, S., 1998. User performance in relation to 3D input device design. *ACM Siggraph Computer Graphics*, *32*(4), pp.50-54.

Zhao, Y.J., Shuralyov, D. and Stuerzlinger, W., 2011, September. Comparison of multiple 3d rotation methods. In *2011 IEEE International Conference on Virtual Environments, Human-Computer Interfaces and Measurement Systems Proceedings* (pp. 1-5). IEEE.