university of groningen

faculty of mathematics and natural sciences

Prop-free 3D interaction with virtual environments in the CAVE using the Microsoft Kinect camera

Master's thesis in Human-Machine Communication

July 2012

Student: Romke van der Meulen, B.Sc.

Internal supervisor: Dr. Fokie Cnossen - Artificial Intelligence

External supervisor: Dr. Tobias Isenberg - Computer Science

ABSTRACT

Immersive virtual environment (IVE) applications traditionally make use of a number of props to support user interaction. In an effort to create an IVE system that is suited to casual use, I combined the Microsoft Kinect, an optical 3D tracking input device that does not require reflective markers, with the CAVE output system. The resulting system supports interaction between the user and the VE without the use of input devices or props.

Using this hardware, I created a number of prototype interfaces. This included two games, an interface to select, move and rotate objects, and an interface for data exploration consisting of a menu system, a technique for zooming in on a point and a technique to simultaneously translate, rotate, and scale the environment. I performed an informal evaluation of this prototype by asking 11 test participants to use the various interfaces on a number of different tasks. The participants were quick to learn how to use the interfaces, and were enthusiastic about their possibilities.

The main drawback of the prototype was that it was prone to tracking errors that disrupted interaction. These errors may be corrected in the future by use of signal analysis and multiple Kinect devices rather than one. The prototype proved to be a successful demonstration of prop-free interaction, and could, once reliability is improved, be used in a range of applications, including data exploration and education.

Figure 1: The CAVE-Kinect prototype in action. The user, without any physical prop in hand, is pointing out a location in the virtual environment, which is illuminated by a virtual flashlight.

ACKNOWLEDGMENTS

Thanks to Frans, Laurens and Pjotr from the Visualization department for their constant help and feedback.

Thanks to Tobias Isenberg for always keeping me on track, and to Fokie Cnossen for helping me dream up new possibilities.

And thanks to my family for teaching me to always do my own thing.

CONTENTS

1

INTRODUCTION

As the operation is ready to begin, the surgeon waves her hands over the patient. Above the patient appears a three dimensional projection of the patient's body, at the same scale as the patient himself. With a wave of her hand, the surgeon makes the skin on the model disappear. The muscles, ribcage and lungs are next, until the heart is left uncovered. With another gesture, she grabs hold of the model around the heart. Pulling her hands apart, the surgeon enlarges the model until the heart is now the size of the operating table. She turns it around until the valves are turned towards her. Another gestures, and a virtual menu appears. Simply by pointing, the surgeon selects a model of the graft that is to be put in, then using both hands, she rotates the virtual graft onto the virtual heart. As the surgeon required no special clothing and did not need to touch any objects to perform these actions, her hands are still sterile. So now that the final simulation is complete, the actual surgery can begin.

More and more ways are emerging for people to experience three-dimensional virtual worlds. Passive ways of seeing such worlds are already becoming common, such as watching a 3D movie at a cinema. However, active interaction in three dimensions between a person and a virtual environment is not yet common outside a small group of specialists.

The natural way for human beings to interact with the real world has always been by using all three spatial dimensions. If a virtual environments is to emulate the real world as closely as possible, it must be represented in three dimensions. The natural way for people to actively interact with such three-dimensional virtual worlds is also by using three dimensions for the interaction. To accommodate natural means of interaction between humans and virtual environments, we need to find ways of supporting 3D interaction.

3D interaction has numerous potential applications in education, simulation, technical design and art. To benefit fully from such applications, 3D interaction must be made accessible for novices, first-time and one-time users. 3D interaction must not only be made reliable and rich in features, but also learnable, usable and available for casual use; i.e. be so natural in use and so free of barriers and special requirements that even a newcomer can begin to interact with the system without having to give it a second thought.

A number of methods of interaction between users and virtual environments have already been explored. Many approaches support 3D interaction through the use of motion tracked props (e.g., Keefe et al., 2001) or worn input devices such as the Data Glove (e.g., LaViola, 2000). Such artifacts have a number of advantages: they are reliable, they allow for great precision in command input, and they make the functionality of the interface tangible to the user.

However, the use of specialized equipment also makes it more difficult for new users to start using the interface: they need to pick up and put on these new input devices; the devices need to be charged, connected and calibrated; all these actions are barriers to casual interaction. To bring 3D interaction to a broader audience, users need to be able to start interacting, even if in limited ways, with 3D virtual objects as quickly and easily as with real ones. Therefore we cannot always depend on worn input devices, interactive surfaces, motion tracked props, or any other required artifact that keep a user from interacting immediately and naturally with virtual environments.

One possible solution for this problem is optical motion tracking of the body of the user itself, without any kind of artifact worn by the user. The Microsoft Kinect (Rowan, 2010), shown in figure 2, may provide such capabilities. The Kinect is an optical input device with one color camera and one IR projecting/reading depth camera. The Kinect provides the capability of reconstructing and tracking user "skeletons"; that is, the camera-relative x, y, z-coordinates of joints such as shoulders, hands, hip, and feet. No special reflective clothing or per-user calibration is necessary: once the system is set up, a user can immediately start interacting.

An interesting property of the Kinect as an input device is that the input is continuous from the moment the user's position is first acquired by the Kinect until the moment it is lost. Furthermore, there is no discrete action that the user can take that is recognized by an off-the-shelve Kinect. This means that by default there is no method available for the user to switch between idle mode and command mode. While with keyboards and mice commands to the system are only issued when a button is pressed, there are no similar events in the Kinect input stream, and some other way of switching modes needs to be introduced. This presents an interesting challenge to overcome in building an interactive system using the Kinect, and is discussed in more detail in section 3.4.

Figure 2: The Microsoft Kinect. The middle lens is for a RGB camera that offers an 8-bit video stream of 640 x 480 pixels. The left lens holds an infrared laser projector and the right a monochrome IR sensor, which together give an 11-bit depth stream of 640 x 480 pixels.

If the Kinect is a suitable input device for casual 3D interaction, we still require an output system that supports casual use. The CAVE (Cave Automatic Virtual Environment), shown in figure 3, may be suited to this task. The CAVE is an immersive virtual environment output system, where projectors display images on four walls of a rectangular room. CAVEs are also capable of stereoscopic display which, combined with head tracking, allows users to see virtual object as if they were floating in space inside the CAVE.

Unlike a head-mounted display, the CAVE can be accessed immediately by casual users, simply by walking inside. Unfortunately some form of tracking, like a cap with reflective markers, and special glasses for 3D display are still required for the stereoscopic display. In the future, we may be able to use a Kinect camera or similar prop-free device to provide head tracking, removing one more obstacle to casual interaction. If a method of autostereoscopy can be added to such a system, we will finally have a walk-in, prop-free, immersive 3D virtual environment.

Given the possibilities and constraints of the Kinect and the CAVE, we need to investigate what kind of human-machine interaction is possible in a CAVE-Kinect based interface. In this thesis I describe my efforts to establish which types of *interaction techniques* are suitable for such an interface. An interaction technique is any form of operation that a user can perform and the system can recognize, like moving and clicking a mouse, typing on a keyboard, but also using voice commands or gestures. I will also address the advantages and drawbacks of a CAVE-Kinect system, and determine whether it allows for prop-free, casual interaction.

Figure 3: The Cave Automatic Virtual Environment (CAVE) at the Electronic Visualization Laboratory of the University of Illinois in Chicago.

In chapter 2 I will examine work previously done using the CAVE and using the Kinect. Most previous systems using the CAVE that allowed for interaction made such interaction available through the use of props. By using the Kinect, I will create a new type of CAVE interaction that does not require props. The Kinect was released only two years ago, and not many projects using the Kinect have been published. One of the uses to which the Kinect has already been put is to create an interface for the operating room that can be operated without touching anything, preserving sterility.

Through iterative phases of design and testing, I have implemented a prototype system using the tracking data from the Kinect and the output capabilities of the CAVE. I created an interface that uses postures and gestures to trigger mode changes, and which supports selecting and moving objects, zooming in and out on a specific point, free manipulation of the viewpoint and performing symbolic system control through a menu. The design details and considerations for this prototype will be discussed in chapter 3.

Two factors made the accuracy of the tracking input less than ideal: first, that the mapping of Kinect coordinates to CAVE coordinates was not completely accurate, and deviated in some parts of the CAVE; second, that the Kinect was placed at a somewhat awkward angle and, more importantly, could only track non-occluded body parts. The first problem may be solved by a more careful calibration of the Kinect's native coordinate system to that of the CAVE. The latter problem may possibly be corrected by the use of multiple Kinects at different angles. These limitations are discussed in more detail in section 3.10.

I have informally evaluated the design of this interface by asking a number of biology students to use the interface to explore a 3D anatomical model of a frog. Most participants were able to quickly learn to use the interface, and were enthusiastic about the possibilities it offered, although most participants also agreed that the lack of accuracy was a major drawback. Using a Likert-scale questionnaire, the participants evaluated the prototype on a number of criteria such as usability, learnability, comfort, *presence*, and fun. Presence (Lombard and Ditton, 1997) is a state where the user ceases to be aware that he or she is standing in a simulator, feeling rather like they are present in the virtual environment. The evaluation and the results are described in depth in chapter 4.

The feedback of this evaluation was used to improve the design of the interface, as well as gain insight into the possibilities and drawbacks of a Kinect-CAVE interface, as discussed in chapter 5. From the feedback, as well as study of users as they were using the interface, I conclude that prop-free interaction is conducive to unrestrained data exploration by users. The prototype interface I've created could already be used in some applications, such as education. There remain problems to solve, foremost of which is the system's inaccuracy. However, the Kinect-CAVE interface has more than succeeded as a demonstration of the possibility and advantages of prop-free 3D interaction.

2

RELATED WORK

In the introduction I have set myself the goal of supporting prop-free, casual interaction with virtual environments in three dimensions. I considered the Microsoft Kinect as an input device, and the Cave Automatic Virtual Environment (CAVE) as an output device.

In this chapter, I will look in more detail at these two devices. I will describe systems previously created using these devices, and discuss whether such systems can be considered available for casual use; i.e. whether a user can, without a second thought, enter the system and begin using it. Where previously designed systems are not available for casual use, I describe how my own approach will differ to create a system that is ready for casual use.

2.1 WORK DONE ON THE CAVE

Figure 4: xSight HMD by Sensics, Inc.

There are two main directions in immersive virtual environment output systems (Brooks Jr, 1999). One is the head-mounted display system or HMD (Fisher et al., 1987), the other is 3D projection surrounding the user, of which the "Cave Automatic Virtual Environment" or CAVE (Cruz-Neira et al., 1992, 1993b) is an example.

The head-mounted display system, of which figure 4 shows an example, consists of a system mounted on the user's head, displaying stereoscopic projections on two displays in front of each of the user's eyes. Sophisticated HMD systems also track the user's head position and angle, so that the projected images can be changed as the user looks around.

The CAVE consists of a room-sized cube with three to six projection screens, each of which displays stereoscopic images that are filtered by a set of shutter glasses that the user wears, creating a 3D effect. The position and orientation of the user's head are tracked. The user can walk around inside the CAVE, and see virtual objects as if they were floating in mid-air, and look at them from different angles, as the projection is adapted to the user's gaze direction.

HMD has a number of advantages over the CAVE. It is smaller, and more mobile. The device can come to the user, where with the CAVE the user must come to the device. Furthermore, the HMD allows the user to see the virtual environment all around, where usually with the CAVE only certain viewing angles are supported by the projection surfaces.

One disadvantage of the HMD is that it completely blocks the real world when the user puts it on. Recent developments allow the user to watch both physical and virtual worlds simultaneously in a technology that is collectively known as augmented reality (e.g., Azuma and Bishop, 1994). However, traditionally one of the advantages that the CAVE holds over HMD is that the shutter glasses used in its 3D projection still allow the user to perceive his or her own body in addition to the virtual environment. Another advantage that the CAVE holds over HMD is that multiple users can enter the virtual environment at once, although often only one of these is head tracked.

Common usage of the CAVE includes product design and exploration of data visualization (Cruz-Neira et al., 1993a; Bryson, 1996) as well as psychological research (Loomis et al., 1999).

CavePainting (Keefe et al., 2001) is a representative example of the use of the CAVE as a VE output device in an interactive system. It also shows the advantage of a user being able to see his or her own hands in addition to the virtual environment. In this application, users can create 3D brush strokes within the CAVE area to create 3D paintings. By seeing their own hand in addition to the virtual environment, painters can adjust their movement to create the brush stroke that is desired. In addition, the CAVE walls were made part of the CavePainting interface, creating a link between physical and virtual environment. The user could 'splash' virtual paint onto the CAVE wall using a bucket, or 'dribble' paint on the CAVE floor.

For its interface CavePainting makes use of a number of physical, motion-tracked props. One is a paint brush, with an attached button that the user can press to begin and end a brush stroke. Several stroke types are available, which the painter can activate by 'dipping' his brush in one of a number of cups on a small table near the CAVE entrance. Audio feedback is given to indicate the changed stroke type. The size of the stroke can be adjusted using a knob on the table or using a pinch glove worn on the non-dominant hand. The use of physical props creates haptic feedback for the user, which Robles-De-La-Torre (2006) shows to be an important factor in creating VE systems that feel natural and immersive. Evaluation of the CavePainting interface by art students also showed that novices could quickly learn how to work with the prop-based interface.

The success of CavePainting's prop-based interface is probably due to the close analogy between the task performed in the virtual environment and a similar task (painting) in a physical environment. Because of this, the properties and affordances of the physical props associated with the task in the physical environment can transfer to the same task in a virtual environment. The same approach may not be as effective in creating an interface for tasks that can only be accomplished in a virtual environment, and have no physical analog.

More importantly, CavePainting is not an interface built for casual use. Users need to pick up one of the props to interact with the system, and need to walk toward the CAVE entrance and perform a physical act to change options for their interaction (brush stroke type/size). This provided the great advantage of making the interaction natural and learnable in a task that was not intended for casual use anyway. However, my goal for the current project will be to create an interface

that the user can begin using the second he or she walks into the CAVE, simply by moving hands or other body parts.

Other systems created using the CAVE follow similar patterns as the CavePainting system, or are even less conducive to casual use. My own approach will be to create an interactive system in the CAVE that can be used without any props, so that the user can interact with the system immediately upon entering the CAVE, no further actions required.

2.2 WORK DONE ON THE KINECT

Traditionally, most user tracking technology has relied on some kind of prop or another. This usually provides the system with good input accuracy, and can improve the interface by providing haptic feedback. However, they also prevent user's from immediately using the interface, offering barriers to casual interaction.

However, recently a number of input technologies have been developed that do not require the user to hold or wear any special prop. Such technologies are called 'touchless', and Bellucci et al. (2010) give a review of a number of these technologies. Starner et al. (1998) show an early example of a vision-based touchless approach. They created a system using a camera mounted on a desk or on a cap worn by the user that could recognize a 40 word vocabulary of American Sign Language with 92 and 97 percent accuracy, respectively, while the user didn't need to wear anything on their hands. More recently Matikainen et al. (2011) used two camera's to recognize where a user was pointing in 3D space by tracking the pointing motion and determining the final pointing angle in two planes. In their conclusions, they considered that the Kinect would soon make a prop-free depth sensor available, which would be beneficial to their goals.

The Kinect, an optical, depth-sensitive input device developed primarily for the gaming industry by Microsoft, can fully track up to two users simultaneously, using feature extractions on depth data to reconstruct the position of 20 separate joints, working in all ambient lighting conditions. Since the Kinect allows for 3D input, can easily be set up and is very affordable, it is currently also finding its way into applications other than gaming. However, as the Kinect has only been publicly available for the last two years, not many of these applications have published results yet. I will discuss some of the preliminary findings and pilot projects.

The Kinect can create a stream of 640 x 480 pixel depth information by projecting infrared light and then measuring that light through a sensor. Stowers et al. (2011) tested whether a depth-map created using the Kinect depth sensor would be sufficient to allow a quadrotor robot to navigate a space, and concluded that the sensor was capable of operating in dynamic environments, and is suitable for use on robotic platforms.

The Kinect offers the possibility of interacting with computer systems without needing to touch a physical controller, retaining sterility. One of the most common applications to which the Kinect has been put, therefore, is in the operating room. Gallo et al. (2011), for example, describe an interface using the Kinect that allows medical personnel

to explore imaging data like CT, MRI or PET images. This interface can be operated using postures and gestures, and makes use of an *activation area* where users must stretch their arm over more than 55% of its length to effect a state transition, preventing unintended state transitions and reducing computational complexity.

My own approach will be somewhat dissimilar. Where the interface from Gallo et al. (2011) was geared toward the manipulation of 2D images, I shall attempt to create an interface that operates on 3D models, and where their interface was operated at a distance, I will create an interface where the work area of the user overlaps the data being operated on.

2.3 SUMMARY

I have introduced the CAVE, an immersive virtual environment output system that has existed for nearly two decades, and the Microsoft Kinect, which has been released only two years ago. The former offers a relatively unconstrained and easy to use environment for viewing 3D projections, the latter an input system affording 3D input from users without the need of any kind of additional artifacts. It is my expectation that the combination of the 3D input possibilities of the Kinect with the 3D output of the CAVE will create a natural mapping that is intuitive for users. In the next chapter I will describe my efforts of creating a prototype system through the combination of these two technologies.

3 INTERFACE PROTOTYPE DESIGN

In previous chapters I set myself the goal of creating an interface that supports casual 3D interaction between users and virtual environments. I decided to use the Microsoft Kinect as an optical tracking input device, and the CAVE as an immersive virtual environment output device.

Since these two devices have not been combined before, research is needed to find whether a viable system can be created with these devices, and what kind of interaction techniques are most suitable to its operation. To answer these questions, I implemented a prototype system. This system supports the drawing of a 3D stick-figure at the inferred location of the user in the CAVE; an interface for selecting, rotating and moving objects; two small games; and an interface for data exploration that supports zooming, freehand view manipulation and symbolic system control through a menu.

In this chapter I describe how this system was designed, and what considerations led to this particular design.

Primary design and development was iterative. I implemented a new feature of the prototype system, then informally evaluated it by testing it myself, or asking colleagues to test the new feature. The results of these tests were then used to adjust the design of the feature, or rebuild parts of it in an effort to anticipate and prevent usability problems, and make the prototype's interface as intuitive in its use as possible.

3.1 HARDWARE SETUP

The prototype was implemented at the Reality Center of the Rijksuniversiteit Groningen's High Performance Computing / Visualization department. One Kinect was attached above the Reality Center's CAVE system, and connected to a PC running a simple server that serves the skeleton coordinates of one person at a time. A schematic overview of this setup can be seen in figure 5.

3.1.1 *Setup of the CAVE*

The CAVE system in the Reality Center of the Rijksuniversiteit Groningen has four projection screens (left, right, back and floor) and uses a cap with attached reflectors to optically track the user's location and gaze direction. This information is sent to 'osgrc', a custom-built application using the OpenSceneGraph library (Burns, 1998). This application is running on one master computer and six slaves connected over a secure LAN. Together these machines generate the stereoscopic images to be displayed on each of the CAVE screens. To view the stereoscopic images, users wear a set of active 3D glasses. Finally, to preserve the floor projection area, users must either remove their shoes or wear galoshes.

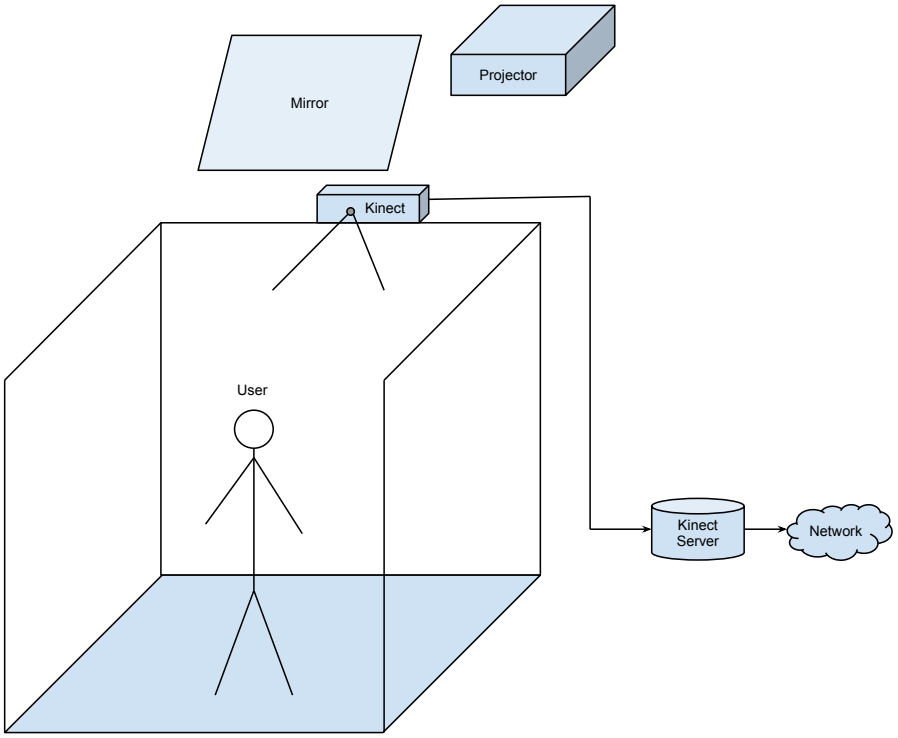


Figure 5: Illustration of the hardware setup for the prototype system. A projector located above and behind the CAVE is aimed at a mirror hanging above the top of the CAVE. Through this, images are projected on the CAVE floor (colored). Similar projectors and mirrors project to the left, right and back wall screens. The Kinect is located above the back screen, aimed downward. It is connected by USB to a PC running a custom-built server application, which transmits the tracking data over the network.

Because the computers controlling the CAVE use the Linux operating system, while the Kinect requires Windows 7, and because these computers are physically removed from the CAVE, a PC running Windows 7 was placed behind the CAVE and the Kinect was connected to it over USB.

I designed a simple server for this PC which accepts TCP connections and continually sends input from the Kinect camera to all connected clients. The server also supports recording input and playing it back, and transforming the coordinate system of the skeleton positions (see section 3.2).

The Kinect can track the positions of up to six people at once. However, since only one CAVE user can wear the reflective cap used in head tracking, an effective maximum of one user at a time was already built into the system. I built the server to only send tracking data of one person over the network, avoiding the additional complexity of having to differentiate data for different users.

3.1.2 *Setup of the Kinect*

The official Microsoft SDK for Kinect offers access to a number of features, including voice and facial recognition, speech recognition

with noise reduction and localization, and skeletal tracking. The skeletal tracking system offers the capability of tracking up to six individuals at once, and for up to two of these users, to extract and track 20 individual joints in 3D space from the depth camera input, as illustrated in figure 6.

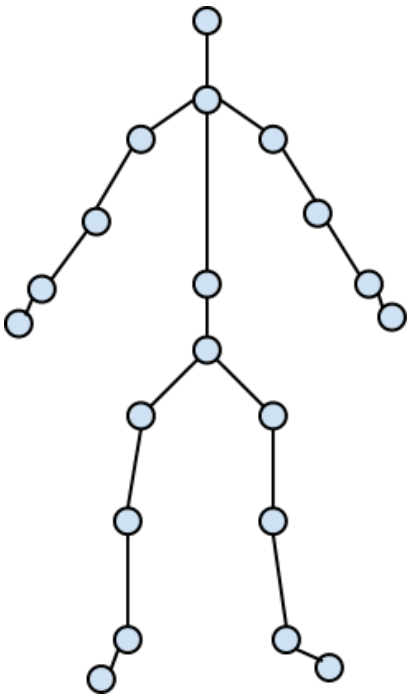


Figure 6: Illustration of joints tracked by Kinect

The default Kinect tracking capabilities can detect the location of a user's hands, but not those of fingers. The Kinect cannot even detect whether the hand is open or closed. A number of open-source solutions have recently been developed for the Kinect, including the openNI framework (openNI, 2011). One application (Rusu et al., 2010) using this framework in combination with the Robot Operating System package (Quigley et al., 2009) uses tracking of a user's separate fingers to develop a virtual touch interface. However, this system expects the user to be in front of the Kinect and at close proximity, and whether the system is usable in the CAVE is unknown. Similar concerns apply to another approach taken by Oikonomidis et al. (2011), who created full hand tracking with a Kinect using a model-based method. It may be possible to develop Kinect tracking that is capable of distinguishing between opened and closed hands, even under the less than ideal conditions of the Kinect-CAVE setup (see section 5.3).

I have used the official Microsoft SDK for Kinect for the development of the current prototype. This did not provide the capability of detecting open or closed hands, nor did I have the opportunity to add such functionality.

For this prototype, I placed a single Kinect sensor in the back of the CAVE, just above the back projection screen. The angle was chosen so as to give the widest tracking area inside the CAVE. Nevertheless, certain areas, especially close to the back projection screen where the Kinect was located, were either outside the Kinect's field of vision or presented the user to the Kinect at such a steep angle as to make tracking impossible. A simple test showed the area of the CAVE that gave reliable tracking to be at least half a meter from the back projection screen, more if the joint to track was close to the ground (see figure 7).

The use of one Kinect camera, rather than several, meant that when the user was turned away from the camera, or one body part occluded another, Kinect tracking became less reliable. This often led to tracking errors.

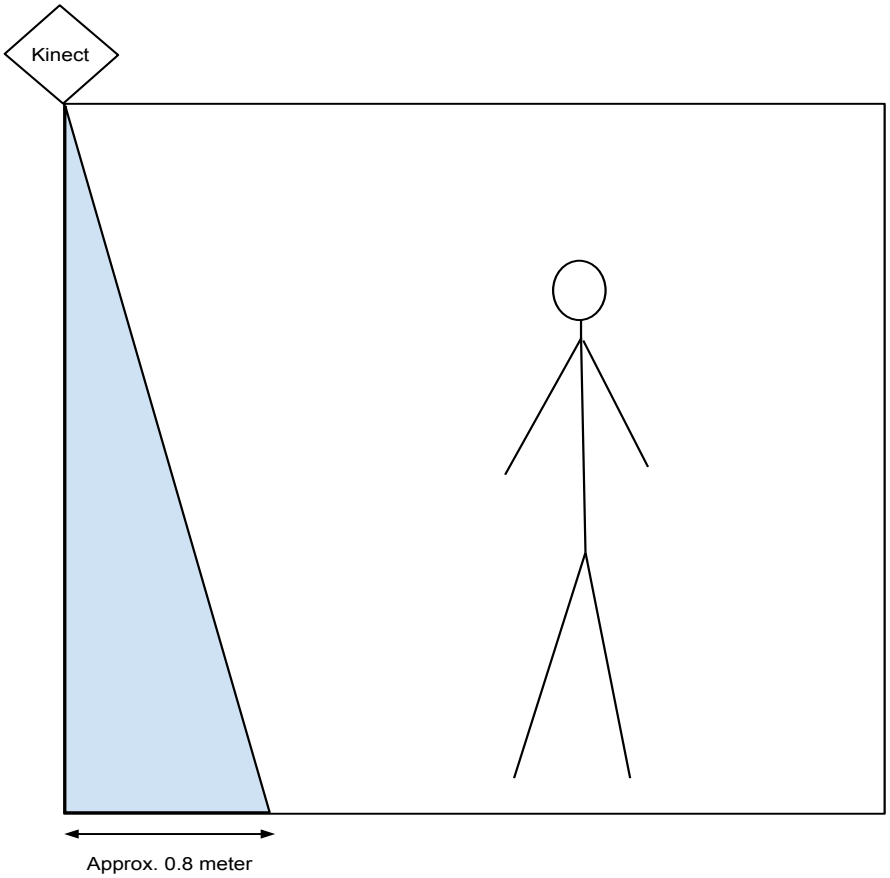


Figure 7: Illustration of CAVE area that can be tracked by the Kinect. The colored section near the back projection wall is an indication of a part of the CAVE where tracking is unavailable. The white space that the user is standing in can be tracked.

3.2 CALIBRATION: MAPPING COORDINATE SYSTEMS

Once the hardware was set up and working correctly, the next problem that needed to be addressed was a transformation of Kinect tracking coordinates to coordinates in the virtual environment. For the user to interact with the virtual environment, the position of the user's joints, particularly the hands, must be correlated, in a single coordinate system, with the position of objects in the virtual world. If the joint's coordinates are known relative to the CAVE, then they can easily be transformed to the coordinate system used by virtual objects by relating the CAVE coordinates to the position and orientation of the virtual camera in the virtual world.

The Kinect skeletal tracking uses a 3D coordinate system relative to the position of the Kinect camera. The x-axis runs parallel to the Kinect sensor bar, the y-axis aligns with the height of the Kinect and the z-axis represents the distance from the front of the camera. To map this coordinate system to that of the user's position inside the CAVE, I developed a simple calibration application, shown in figure 8. To use it, a user stands in the CAVE and holds a tracked joint at a position of which the CAVE coordinates are known. An operator then takes the Kinect coordinates for this joint's position, and maps them to the CAVE coordinates. A number of such mapped coordinates are

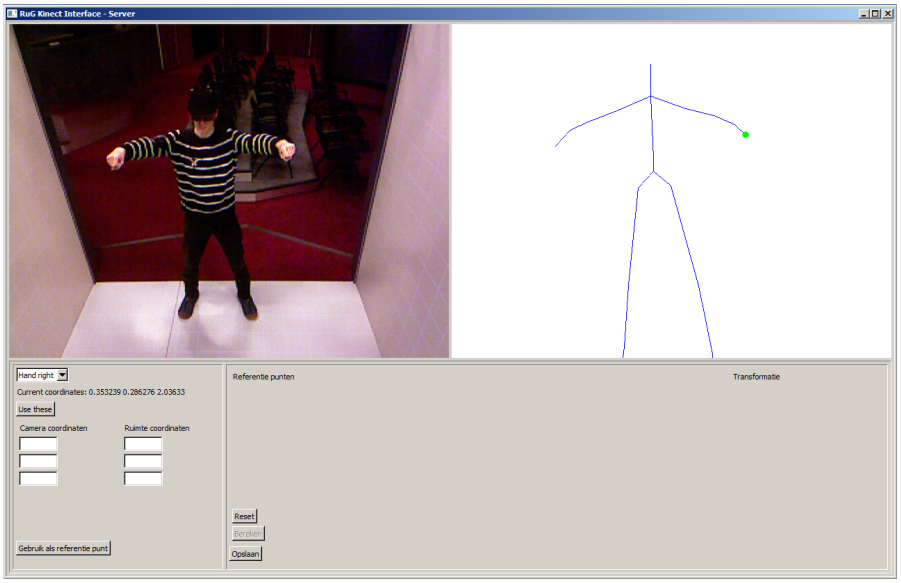


Figure 8: A screenshot of the calibration application, also showing the perspective of the Kinect camera. One person is standing in the CAVE, making a pose as to be clearly identified by the Kinect tracking system. The joint to use for calibration, in this case the right hand, can be specified in the bottom left panel and is indicated with a green dot in the top right panel, which shows the current tracking data (horizontally inverted). By clicking the 'use these' button, the current tracking coordinates are entered in the left column of three input boxes. The person responsible for calibration can then enter the corresponding CAVE coordinates in the right column and press 'Use as reference point' (Dutch). After four reference points have been entered, a mapping is generated and displayed in the bottom right panel. After pressing 'save', this mapping is written to a file, where it can later be read by the Kinect data server.

then used for the *estimateAffine3D* algorithm of the openCV library (Bradski, 2000) to estimate a single transform matrix using random sampling. This transform only needs to be calculated once, and is then applied by the Kinect server to all Kinect tracking data before sending it to client applications.

After the first calibration, it became evident that the mapping was still somewhat inaccurate: the calculated position of the user's hands were often off by as much as 20 centimeters. I manually adjusted the transform matrix by adding additional corrections to scaling, rotation and translation, until the projected joint locations (see also section 3.3) were more accurate to the actual joint locations. The mapped user joint coordinates then proved to be accurate to within about five centimeters for much of the CAVE area, but distortion still occurred within areas of the CAVE that were near the left and right projection screens. For present purposes the mapping accuracy sufficed, but with more time and effort the transform matrix could be made to map Kinect input accurately throughout the CAVE.

Once the Kinect was set up and the server was transforming Kinect coordinates to CAVE coordinates, I implemented a simple visualization of the mapped coordinates, to test the mapping and also to gain more insight into the possibilities and restrictions of Kinect tracking in the current setup. I created a simple 3D stick figure in the CAVE at the coordinates where the user was calculated to be. A small sphere was drawn at the location of each tracked joint, and cylinders were drawn between them to indicate their position in the user's skeleton. If the location matched exactly, the user's body would occlude the projection, but since the location was usually off by a few centimeters, the result seemed to hover close to the user's limbs. I therefore called this projection the 'exoskeleton'. Figure 9 shows this system in use.

The exoskeleton proved to be a useful tool in discovering the possibilities and limitations of the Kinect input. For example, when the position of the left hand as determined by the Kinect suddenly started to vary because it was occluded by the right hand, the exoskeleton showed a fast jitter that warned that the current pose could present difficulties if used in the interface. By trial and error, I used this system to come up with a number of reasonably reliable and natural poses, which I later used in the various interfaces.

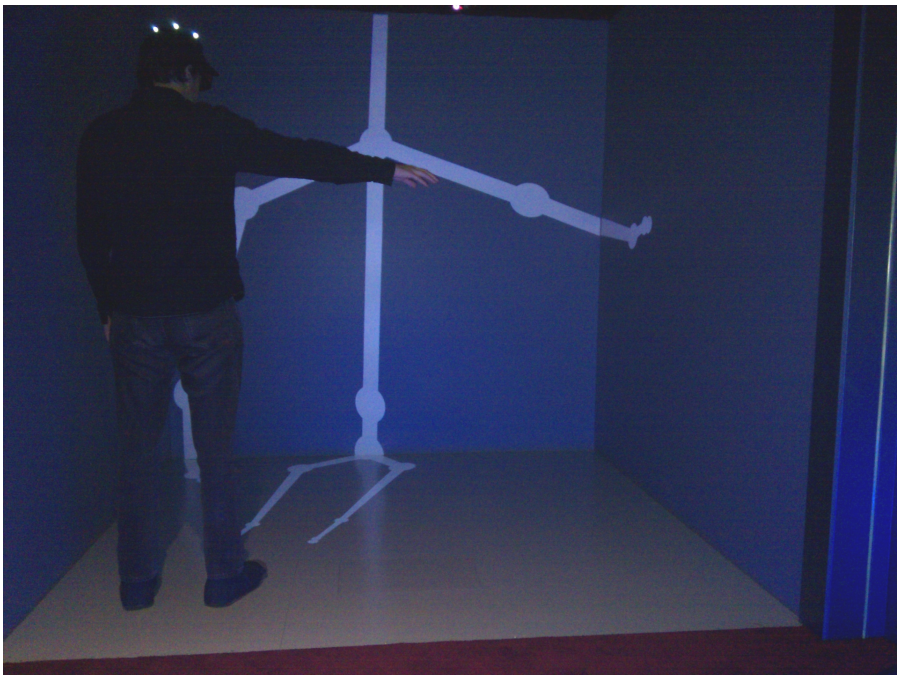


Figure 9: A user in the CAVE with the exoskeleton being displayed. The exoskeleton location, which normally matches the user's, was offset to the front and right for this picture to make the exoskeleton clearly visible. CAVE stereoscopic display was disabled for this picture.

Before I could begin to design actual interfaces using the Kinect's tracking data, I encountered the most difficult problem that such a system would have to solve.

Traditional input media such as keyboards and mice have one great advantage: it is instantly clear whether the user is currently using the input device to input commands. The user needs to press a button if some action is desired, and as long as no button press is detected, no user command is detected.

This is not the case with input from tracking devices such as the Kinect: as long as a user is in range, a continuous stream of input is given. This stream represents both meaningless behavior, when a user does not wish to interact with the system, and meaningful behavior, when the user wishes to execute some action. This presents the problem of trying to separate, as data is coming in, which parts of the input stream together form a single user command and parameter set, and which parts of the stream ought to be ignored.

I call this the 'user intention problem' – the problem of trying to pick meaningful input from a continuous stream composed of both casual and deliberate behavior. The problem can be split into three distinct parts, listed in table 1.

The first, and arguably the most difficult part, consists of separating command input from meaningless input. This separation must occur as data is coming in. The dimension to segment here is time: at what moment in time does the input go from being meaningless to representing considered behavior from a user trying to give a command? When input quality is low, errors may occur in the segmentation, which results in commands being missed when given, or detected when not present.

The second part occurs once we know that the user is trying to input a command. The question then becomes: which command? An option needs to be chosen from a limited symbolic set of available commands, optionally limited further by the context in which the command is detected. When input quality is low, it becomes more difficult to accurately classify the input as a command, causing incorrect commands to be chosen.

Problem Description	**Problem Type**	**Dimension**	**Input quality affects...**
When does the user intend to give a command?	Segmentation	Time	Fidelity
Which command does the user intent to give?	Classification	Symbolic	Accuracy
What are the parameters for the command?	Quantification	Space	Precision

Table 1: Description of the three components of the user intention problem

The third part, finally, becomes relevant once the command to execute has been classified. Most commands will require parameters to be specified. A command to zoom in, for example, will need to know where to zoom in on and how far to zoom in. The user will need to specify these parameters. This can be done symbolically, e.g. by entering an amount on a virtual keyboard, but it may be more intuitive for users to specify the amount using their own body in space, e.g. by holding there hands further apart or closer together. In this case, low input quality will affect the precision with which parameters can be specified, and errors will result in incorrect values being used with the command.

3.4.1 *Using triggers to change modes*

The problem of temporally segmenting the input stream into commands and casual behavior can be solved by introducing triggers. A trigger is any kind of event that the user can produce and that can be recognized from the input stream, signifying that the behavior before the trigger should be seen as different from the behavior after the trigger. By presenting a trigger, the user can cause the system to enter a different mode.

These triggers can be part of the same input stream they segment, but this is not required. For example, the Kinect tracking input stream can be segmented by pushing a button on a Wii-mote or by the user giving voice commands in the style first explored in Bolt's famous 'put-that-there' interface (Bolt, 1980). However, adding an artifact like a remote control to the interface would undermine my efforts to create a prop-free interaction method. And although voice commands would not have added additional props to the interface, they would have presented an additional modality and a range of interface options that needed to be learned before being useful, decreasing the value of the system for casual use. Furthermore, speech recognition in Dutch was not available at the time the prototype was built. I therefore did not pursue the use of voice commands as triggers.

This left me with triggers based on Kinect tracking data. The only types of trigger that could be used with this data were gestures and postures. Gesture detection, especially in 3D, added a number of additional challenges, such as classifying gestures reliably and accurately. I considered the use of existing technologies for gesture detection, using Hidden Markov Models (e.g., Keskin, 2006), other machine learning techniques (e.g., Lai et al., 2012) or using simple geometric templates (e.g., Wobbrock et al., 2007). However, some of my own early experimentation with posture detection showed it to be far more easy to realize, and the use of postures rather than gestures showed no obvious disadvantages.

I have therefore primarily made use of postures and timing[1] to create triggers. For example: if the user assumes a posture where both hands are held at least 40cm in front of the body (the 'back' joint in the skeleton data) and at least 80 cm apart, and then holds this pose for at least one second, the free view manipulation interface is triggered. I have only used one gesture, namely clapping hands together and moving them apart again, as a trigger for the menu.

As previously explained, when parts of the user were obscured to the Kinect, tracking errors occurred. In such cases, the Kinect makes a best guess as to the location of the obscured joints. This could lead to misses in posture detection, especially if a posture involved body parts overlapping each other. It could also lead to false positives, when the estimated location of an obscured joint met the criteria for the detection of some posture that was not actually in use.

For the user to maintain a correct mental model of the system state, the user must be able to determine when a particular trigger is detected. Providing feedback with these triggers proved to be very important. Not doing so could lead to mode errors (Norman, 1981), one of the most persistent types of usability problems.

I made use of visual feedback in this prototype, giving each interaction technique an interface of virtual objects drawn inside the CAVE. This allowed users to repeat a posture in case it was not detected, or cancel an interaction when a posture was falsely detected. I introduced a central 'cancel' posture that could be used at all times to cancel all ongoing interactions and return the system to the idle mode by holding both hands some distance behind oneself.

Which postures a user can use depends on what the user is currently doing. For example, when the user is inputting a parameter which is determined from the positions of the hands, the hands can not be used in any kind of posture or gesture that changes the position of the hands. In this case, I have found the only natural way of ending such an interaction to be by having the user keep both hands still for a predetermined time.

3.5 OBJECT MANIPULATION

In the first design phases, I focused the development of my prototypes on supporting operations relevant to the uses to which the Visualization department's CAVE was traditionally put. This included large scale models, hundreds of meters long, which were often architectural in nature. A common operation in these models, and one I tried to support with my new interface, was that of altering the orientation an position of objects in the virtual model, e.g. moving a chair inside the model of an office but also moving entire buildings on a model of a terrain.

1 I often used poses that needed to be maintained for a certain period before being unambiguously identified. Since no actual motion is involved, I refer to such poses as postures, although it might also be argued that any interaction technique that involves time as well as a certain pose should be referred to as a gesture. For the rest of this thesis, I will use 'gesture' to refer to any behavior by the user involving motion, and 'posture' for poses that are held still, even if timing is involved.

3.5.1 *Selection*

The first step in object manipulation is the selection of the object, or objects, to manipulate. Selection consists of two steps: indicating the object to select, and confirming the indication for selection. Both steps require clear feedback to prevent accidentally selecting the wrong object, or making a selection without being aware of it.

The tracking errors that arose from the imperfect mapping of coordinate systems, as well as tracking errors due to joints being obscured to the Kinect, made it impractical to work with objects that were not within physical reach of the user. Interacting with such distant objects would require some form of pointing technique, e.g. ray casting or arm extension (Bowman and Hodges, 1997; Poupyrev et al., 1996), or scaling technique like World In Miniature or WIM (Stoakley et al., 1995). Since ray casting would have been unreliable with these tracking errors, particularly at longer virtual distances, and working with dense models like a WIM would require millimeter scale accuracy that the Kinect could not provide, I have focused on only selecting objects using direct indication, and ignored pointing and scaling techniques. To select distant objects, the user would have to travel to bring the object within arm's reach.

The simplest form of direct indication is to hold one's hand on the surface of or inside the object to select. However, a user may accidentally hold a hand inside an object while walking through the CAVE, without intending to make a selection. In trying to find techniques to prevent such accidental selections, I came upon two possible solutions.

If a selection were not made immediately upon indicating an object, but only after an object was indicated consistently for a certain period, accidental selections would become less likely. The basic operation for the user to perform would change little, and if proper feedback was applied to both object indication and selection, the user could maintain an accurate grasp of system state.

Another natural way for users to indicate an object for selection would be to grasp it with both hands, as if to pick it up. Such a posture would reduce the chances of a selection being made accidentally.

I created three selection mechanisms to compare each of these approaches:

- Touch
- Hold
- Enclose

In the case of 'touch', a virtual object becomes selected as soon as a user intersects it with a hand. This has the advantage that no additional trigger is needed, making this technique easier to use and shortening its temporal span. However, the drawback to this technique is that it is very easy to get a false positive. If no additional operations occur automatically after an object is selected, the problem of false positives is slightly lessened, but the user still runs the risk of losing an old object selection when accidentally making a new selection.

The 'hold' technique again requires the user to intersect the object with either hand, but this time the object is only marked as indicated, and the user must keep their hands inside the object for one second before the object is marked as selected.

The 'enclose' technique uses a posture with which the user can indicate an object by placing both hands on either side of that object. The object is then marked as indicated. If the user maintains the posture for one second, the object is marked as selected.

3.5.2 *Moving and rotating objects*

Once an object is selected, it can be moved around and rotated within the confines of the CAVE. Since moving was the only operation on a single object that was implemented in the prototype I developed, I did not add another trigger to enter the mode for moving or rotating: once an object was selected, it would immediately trigger the object moving/rotating mode.

There were two design considerations for moving objects: whether to preserve the initial angle at which the object is grabbed or snap the orientation of the object to that of the user's wrist; and over which axes to allow rotations: only heading (the direction the arm is facing) or also pitch (whether the wrist is pitched up or down). Roll was not a candidate, as the roll of the user's wrist cannot be determined from the Kinect input.

The model for which I implemented this technique consisted of an archaeological site, namely a cave including an entrance to what was once a burial chamber. When the model was lit using the standard lighting, the cave walls did not look as expected. To remedy this aesthetic problem, my colleagues at the visualization department had added a number of custom lights to the model, including a sphere emanating white light, a model of an oil lamp emanating yellow light, and a flash light, casting a more focused beam.

The tracking of the user's wrist's pitch did not prove as accurate as that of the wrist's heading, primarily because the user can change the wrist's pitch independent of the arm, unlike with heading. However, some informal experimentation showed that the added functionality of using both heading and pitch to orient the moved object was quite useful, so both axes were used regardless of tracking errors.

For the flash light, it made sense to have the light beam follow the current direction of the user's wrist, as seen in figure 10, while for the spherical light and oil lamp, the angle of the object to the user's wrist made little difference. We therefore decided to make 'snap-to-wrist', as it was called, the standard way to orient objects while moving them.

Some quick studies outside the archaeological model showed that there are also some advantages to preserving initial angle between wrist and object: this makes it easier, for example, to turn an object fully around, a function nearly impossible in the 'snap-to-wrist' approach as the user would have to turn his or her back to the Kinect to turn the object around, which results in severe tracking errors. There may also be situations where it is advantageous for users to only rotate around a single axis, i.e. either heading or pitch. Future interfaces may choose to offer more than one way to move and rotate objects.

Figure 10: A user moving a flash light. The orientation of the object is directly related to orientation of the user's wrist. The result in this case is that the location the user is pointing at is the location that is illuminated. CAVE stereoscopic display was disabled for this picture.

In the end, object manipulation was one of the last features to be completed (see section 3.8 for an explanation), and was only briefly and informally evaluated. More research may show the full potential of this particular interaction technique.

3.6 GAMES

During design, I sometimes needed to evaluate particular features of an interface before the interface itself could be completed. Examples of such features were the intersection test used to detect when a user is touching a virtual object, or the three selection mechanisms discussed in section 3.5.1. However, these basic features in themselves could not be used in the same task that would later be used to evaluate to complete interface.

In such cases, I created games that allowed me to test these basic features. These games had a number of advantages: they showcased basic features of the system in a way that anybody could understand, and they were inherently more motivating than testing these features on a basic experimental task. In fact, during evaluation these games were often considered some of the best uses of the Kinect-CAVE prototype.

3.6.1 *Pong*

The first system created after the exoskeleton was the Pong game. It was designed to test the Kinect tracking capabilities and also to see if we could accurately test for intersections, i.e. detecting when the user was touching a virtual object.

Two flat, round surfaces were placed at all times at the calculated position of the user's wrists (found to be more stable than the position of the user's hands). These were called the 'bats'. A small sphere represented a virtual ping-pong ball. At the start of the round, shown in figure 11, this ball was placed 1 meter from the CAVE entrance, 1.25 meters above the floor. When the user moved either hand to or through the ball, an intersection of wrist and ball was detected. In that case the game would start by giving the ball a random direction away from the CAVE entrance. The ball started out with low speed, but kept accelerating until the user finally missed it.

If an intersection between one of the wrists and the ball was detected, the ball direction was changed to head toward the back projection screen. The round ended if the user did not deflect the ball as it moved past, getting behind the user. In this case, the ball was replaced at the initial position, and a new round would start. The goal for the user was to keep a round going as long as possible, while the ball accelerated, increasing the difficulty of the game.

If the ball position was found to be near one of the boundaries of the game world, it was reflected back. Initially I chose the CAVE projection surfaces as boundaries, but soon found that it was too difficult for

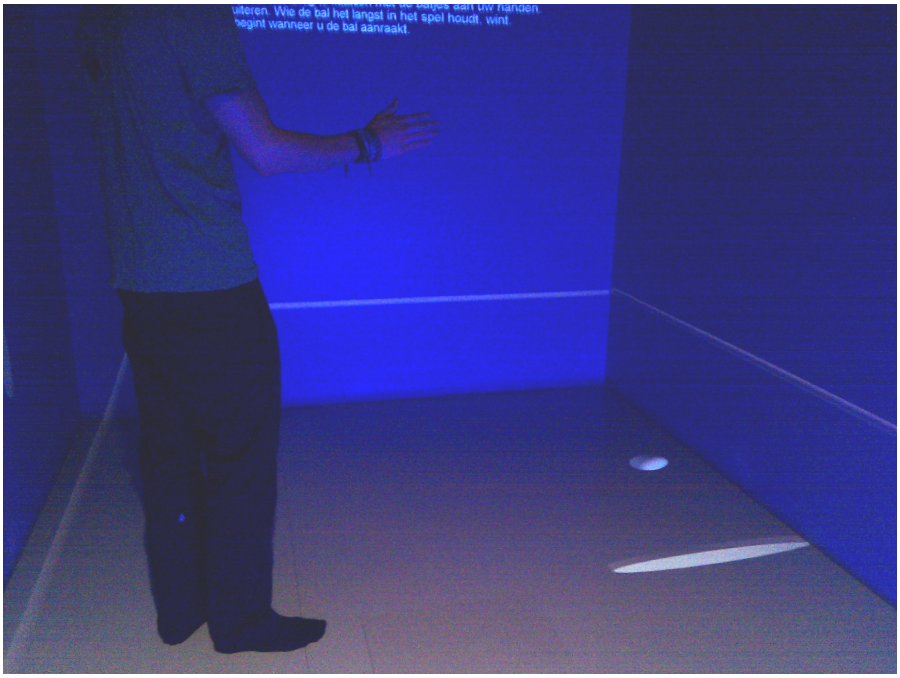


Figure 11: A user playing Pong, just before the round begins. The white sphere is the ball, the orange ellipse is the bat. The orange lines slightly above the floor indicate the boundaries of the game world. CAVE stereoscopic display was disabled for this picture.

an average user to physically reach all locations this allowed. It also frequently occurred that a user slammed a hand into a projection screen in an effort to reflect the ball, or that a user needed to crouch to reach the ball, which led to tracking errors from the Kinect. I therefore narrowed the boundaries in all three dimensions, and projected a number of lines to indicate the edges, as seen in figure 11. It was remarked by several participants that these boundaries made the game quite similar to squash.

3.6.2 *Bubble Burst*

Bubble Burst was created to test the different object selection techniques, which I discuss in section 3.5.1. The game was designed so that at the start of each round, the selection technique could be switched, so that all techniques could be tested during a single game.

The game consisted of six spheres of differing sizes and colors being projected in random locations in the CAVE (within some limits). The object of the game was for the user to make each sphere disappear by selecting it. The spheres had to be selected in a certain order: according to size from small to big or from big to small; or in order of rainbow colors from red to purple or from purple to red.

Each round would start with text projected on the back CAVE screen, explaining which selection mechanism was active and which order the spheres had to be selected in. During the first five rounds, this text would be displayed for ten seconds, with a countdown beneath the text. After five rounds, the user was assumed to be familiar with the game mechanics, and the round explanation was displayed for five seconds.

When the countdown was over, the round began. The text on the back wall was replaced by a short title indicating the order to select spheres in for this round, as well as a timer indicating how long the current round has lasted. The object for users was to clear the round in as short a time as possible.

Also at the start of the round, all spheres appeared at once at their designated locations. The space within which spheres were allowed to appear were parameters of the game. After several tries, I found gameplay was best if spheres appeared at least 65 cm from left and right projection screens, 65 cm from the entrance, and no less than 95 cm from the back projection screen.

The user then needed to use the currently enabled selection technique to select each of the spheres in turn. If the wrong sphere was selected, the selection markings on the sphere were kept in place, while a different marking as well as a short white cylinder indicated the sphere that the user should have selected, as shown in figure 12. The text on the back wall indicated that the round had ended in failure. If the right sphere was selected, it was immediately removed (hence the name 'Bubble Burst'). After the last sphere disappeared, the text on the back wall was replaced with a congratulation on a successfully completed round, along with the number of seconds the round had taken.

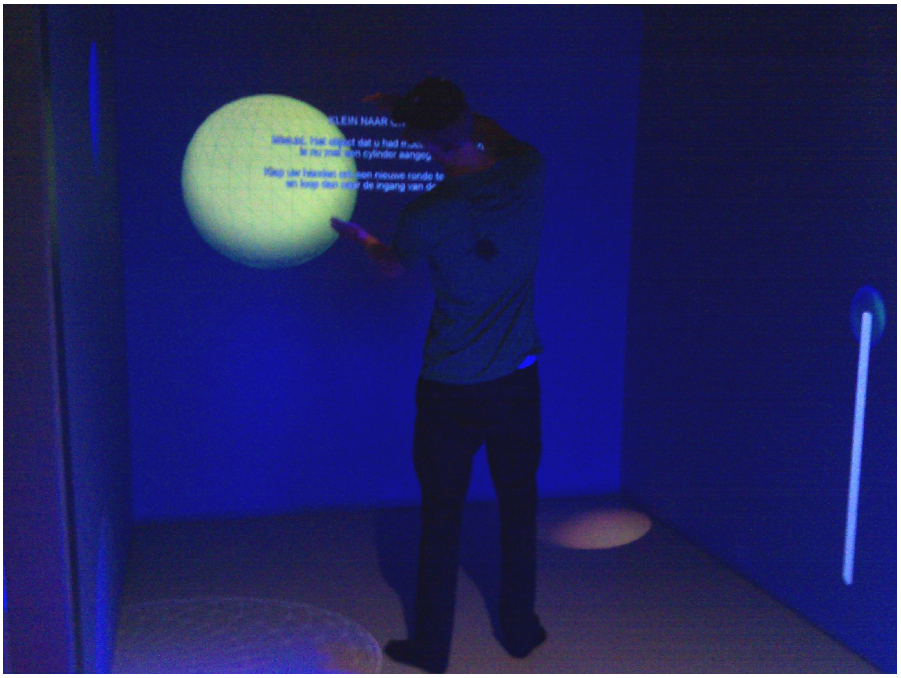


Figure 12: A user playing Bubble Burst. The user is in the process of selecting the yellow bubble in the back using the 'enclose' mechanism: this bubble is marked as being indicated by being outline in green lines. White lines surrounding the bubble in front indicate that this bubble has previously been selected. This was not the correct bubble to select, as the order for this round was small to big. At the right, a white cylinder indicates the sphere that should have been selected. The text displayed over the back bubble indicates that the round has been lost. CAVE stereoscopic display was disabled for this picture.

Whether the round was won or lost, the next step for the user, as advised by the text on the back screen, was to clap his or her hands together to begin the next round. I found that users often accidentally lost a round when they stayed in the same position at the end of the round, and accidentally selected a sphere when the next round began with the user in the middle of the space occupied with spheres. I therefore extended the text at the end of a round with the advice to move back to the CAVE entrance before beginning the next round.

3.7 SWITCHING APPLICATION DOMAINS

After working for some time on object manipulation (see section 3.5), it became clear that the chosen application domain, namely architectural models, did not lend itself naturally to a Kinect-CAVE interface. The reason for this was that these models were quite large, and it was explained in section 3.5.1 that the best use for Kinect input was only manipulation of objects that are within physical reach of the user. Since in large architectural models this included only a fraction of the model, the interface was ineffective.

Several solutions were considered, among them the option of using World In Miniature (WIM) or scaling to bring large models into reach of the user. The problem with this solution was that at such extreme scales, input needed to be precise to within millimeters to successfully manipulate parts of the model, and the Kinect tracking clearly lacked this precision. Another problem would be that switching between scales was likely to cause users discomfort by increasing the chances of simulator sickness.

A better solution was to switch application domains from architectural models to exploration of data visualizations. Such visualizations often consist of a single composite object, which can successfully be scaled to fit within the confines of the CAVE.

The main requirements for an interface for data exploration consist of techniques for manipulating the view of the data, and techniques to access and alter parameters of the visualization. I implemented two different techniques for manipulating the view, which I discuss in section 3.8. I also implemented a technique for symbolic control of the visualization using a menu, which I discuss in section 3.9.

3.8 VIEW MANIPULATION

The most important task for an interface for data exploration is view manipulation. View manipulation allows the user to transform his or her perspective on the data and reveal patterns that may not previously have been visible.

To support view manipulation, I implemented two interaction techniques. One was a technique for zooming in and out on a single point, and the second a view manipulation technique using two points that supports scaling, rotation and translation. I call the latter 'free view manipulation'. This free view manipulation turned out to be a particularly effective tool in data exploration, and no additional view manipulation techniques needed to be explored for the current prototype.

3.8.1 *Zooming*

The most basic way of exploring a data visualization is to look at each part of the data in detail. This can be supported by implementing a zooming technique.

I first designed a technique that simply scaled the entire model up around the origin point of the model. This turned out not to work well, as the scaled world also moved away from the user if the user was not standing exactly at the model's origin point. To prevent such moving, the point to zoom in or out on, i.e. the point around which to scale the model, needed to be chosen with more care.

I decided to allow the user to define the zoom point himself. The complete zoom technique now consisted of the following steps:

1. Trigger the interface with a posture of holding both hands together for at least two seconds above the point in the virtual environment to zoom in on.

2. An axis cross (three perpendicular cylinders), representing the zoom point appears where the hands are held, or if a selected object is found within 65 centimeters of that point, the axis cross and zoom point are placed at that object's center. A white bar (a cylinder between two spheres) appears before the user.

3. The axis cross is red as long as the zoom point is not moving. By touching the cross, it turns green and moves along with the hand used to touch it, as shown in figure 13. When that hand is held still for at least two seconds, the cross turns red again and stops moving. This can be used to fine-tune the zooming point.

4. When the zoom point is set at the desired location, zooming can start by simultaneously touching both spheres of the slider interface. If the slider is not touched within six seconds of the time the zoom point was placed, it is assumed the zoom operation was not intended, and the zoom interaction is canceled. If the slider is touched, the spheres of the slider interface move along with the user's hand.

5. While active, the current distance between the two slider spheres, compared to their initial distance, is used as a measure for the zoom factor. Hands held closer together than the initial distance will zoom out, hands held further apart will zoom in.

6. When both hands are held still for four seconds the zoom action is considered complete, and the zoom interface disappears. Alternatively, the cancel gesture (both hands held some distance behind the user) canceled the interaction and reset the zoom level to the value that was active when the interaction began.

The zoom technique in its entirety took a lot of time for users to complete. I therefore tried to enable the user to specify great zoom distances, so that even for extreme zoom values, only a single interaction was needed. To that end, I used formula 1 to calculate the factor by which to scale the model, where a and b are the position vectors of the left and right slider sphere respectively, $|||b - a|||$ is the absolute length between them, and c is the initial length of the slider. The value of 3.6 was found after some experimentation to allow a great enough range of zoom values, while preserving enough precision in the values closer to the initial zoom level.

$$\frac{|||b - a|||^{3.6}}{c} \tag{1}$$

The first problem found with this technique was that the posture of hands held together conflicted with the gesture of clapping hands together, which was used for the menu (see section 3.9). Although the two could be separated in theory by the time the posture of hands

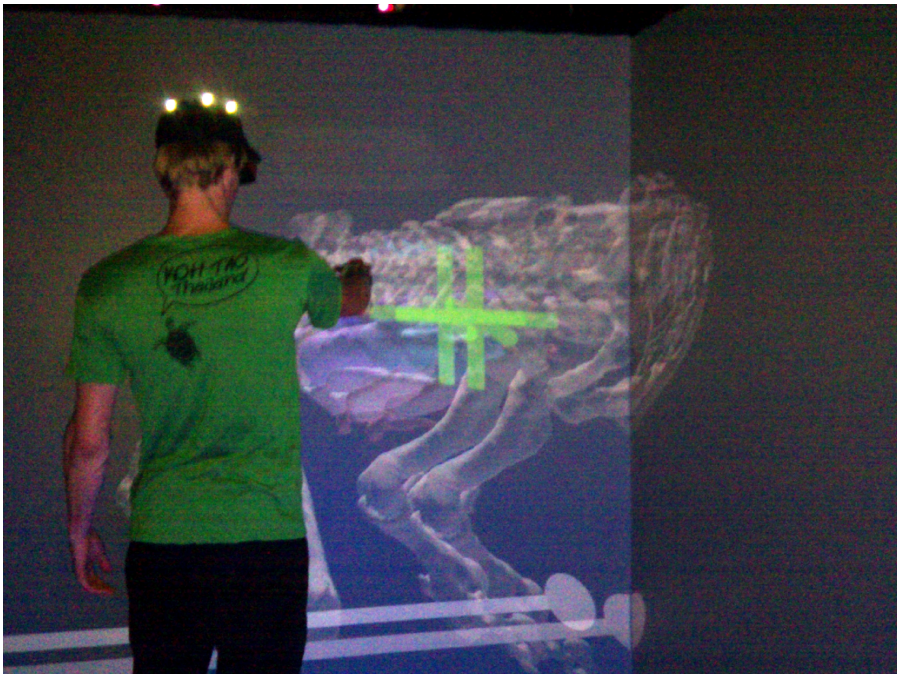


Figure 13: A user using the zoom interface. The user is currently placing the zoom point, which is green as long as it is not in place. The white bar at the bottom, surrounded by two spheres, represents the slider used to input zoom factor. Doubling is caused by the CAVE's stereoscopic display.

together was maintained, in practice it turned out that hands close together were problematic for the Kinect to detect, and tracking errors occurred, so that although the user maintained a posture of hands held together, the system recognized this as a clapping gesture.

To solve this problem, a different posture was chosen. In this case, the chosen posture was to hold one hand to an eye, indicating the desire to look closer, and the other hand held over the location to zoom in on. This posture was changed again during the user evaluation: see 4.7.2.

A problem that was found during informal evaluation by colleagues was that many of the mode switches used in this technique were system-controlled. To add more user control, the timing used to place the axis cross was replaced by a posture: when the hand not holding the axis cross is brought in proximity to the hand that is, the axis cross is placed and turns red. To prevent the user from accidentally picking it up again, a hidden timer was built in that prevented the user from picking up the axis cross for one second after it was placed.

3.8.2 *Free view manipulation*

After a few uses of the zoom technique, it was already becoming evident that defining a zoom point and then a zoom factor was not very intuitive. While looking for more intuitive ways of doing a zoom operation, we considered the 'pinching' gesture used in touchscreen interfaces. The concept with this technique is that a user 'grabs' two points on the screen, and pulls them apart to zoom in or pushes

them together to zoom out. A 3D equivalent would involve the user 'grabbing' the virtual environment in two points, then pulling these two points apart to zoom in or push them together to zoom out. An additional benefit of this technique would be that it allowed the user to simultaneously rotate and translate the model, by moving the grabpoints relative to each other.

The main problem for this technique was how to allow the user to define the two grabpoints. The natural way would be for users to physically make a grabbing gesture with their hands, but this could not be detected by the Kinect. Therefore, triggers would need to be introduced that allowed a user to define the grabpoints before grabbing on to them. A natural trigger to pick up a grabpoint to place it elsewhere would be to simply touch it. However, releasing a grabpoint once it was picked up was a bit more difficult, especially if both grabpoints were being placed at the same time. Since in this case both hands would be engaged, the only trigger I could find was to hold a hand still for two seconds to release a grabpoint.

The initial design for the free view manipulation technique consisted of these steps:

1. Trigger the interface with a posture of holding both hands together for one second (this posture was no longer used by the zooming technique at this point).

2. Two spheres appear, indicating the 'grabpoints'. Initially, they are placed around the location where the posture was detected.

3. The spheres are red while static. By touching either or both, they turn green and move along with their corresponding hand (left sphere with left hand, right sphere with right hand), as shown in figure 14. When held still for two seconds, the spheres turn red again and stop moving.

4. When the grabpoints are placed to satisfaction, the user must lock them in place by using the same posture that was used to trigger the interface in step 1. When locked in place, the spheres turn blue.

5. When the spheres are blue, the user must touch both to begin the zoom action. Before the 'cancel' posture was added, if the spheres were not touched within six seconds, the view manipulation was assumed to have been triggered unintentionally and was canceled.

6. When zooming is engaged, the entire environment is scaled, rotated and translated so that the points in the environment that were the original grabpoints now fall at the current location of the user's hand. This is shown in figure 15.

7. The user can pull the grabpoints apart to zoom in, push them together to zoom out, move hands to translate the entire environment, and rotate the environment by moving the hands relative to each other.

One of the first problems encountered with this procedure was that sometimes the grabpoints were placed too far apart to touch both simultaneously. Even if they were not, tracking errors or poor hand-eye coordination sometimes prevented users from engaging the view manipulation mode. If this went on for six seconds, the view manipulation operation was unintentionally canceled. This latter problem was again solved when the cancel timing was replaced by the cancel posture. The problem of touching both grabpoints simultaneously was solved by engaging the view manipulation mode when either point was touched. If the user had both hands together when this happened, the view manipulation mode would engage with a rather large sudden change, as both grabpoints were suddenly placed close together. Fortunately, users intuitively stretched their hands to both grabpoints, assuming both must be touched. Most users were unaware that touching only one grabpoint would engage view manipulation.

Another problem was that, as was the case with the zoom technique, the posture of holding both hands together conflicted with the clapping gesture used to summon the menu (see section 3.9). This was solved by creating a new posture to engage the view manipulation technique: both hands held in front of the body (at least 40 cm in front of the 'spine' joint) and at least 1 meter apart from each other (later shortened to 80 centimeters to make the posture easier to use). If this posture was maintained for at least one second, the view manipulation interface was shown. This posture turned out to be relatively easy to

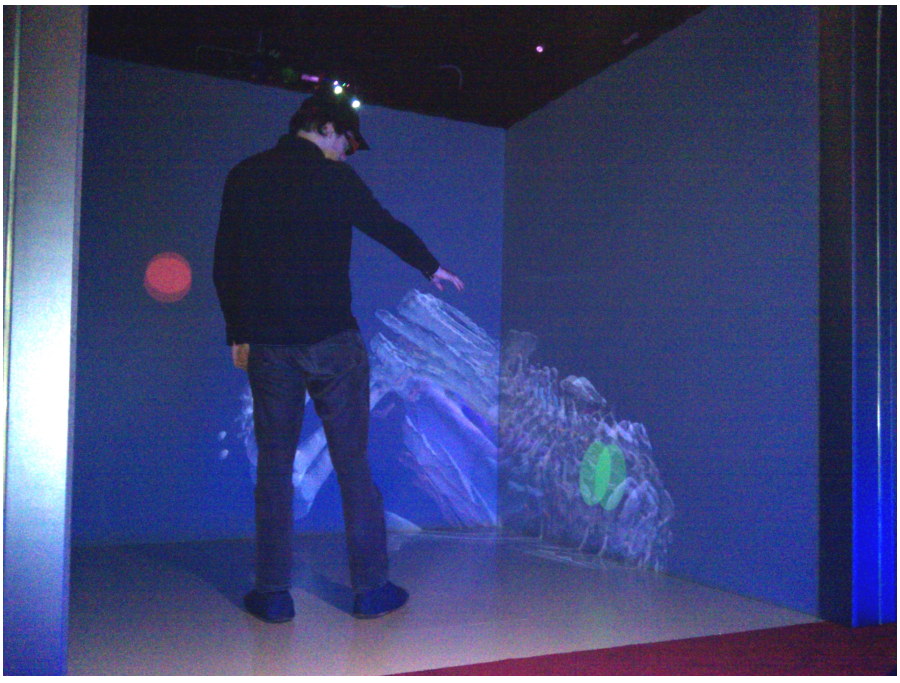


Figure 14: A user initiating the free view manipulation interface. The user is placing the right grabpoint. It is colored green to indicate that it is being moved. The other grabpoint is red, indicating that it is not being moved and has not yet been locked. Doubling is caused by the CAVE's stereoscopic display.

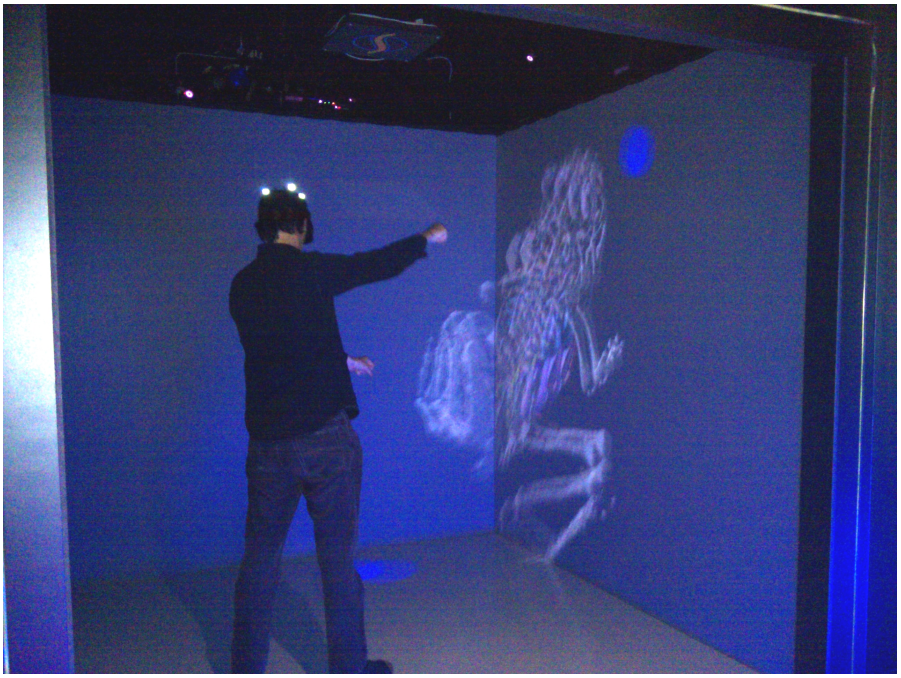


Figure 15: A user using the free view manipulation interface. The two grabpoints are blue, indicating that they are locked in place. The user is currently rotating and translating the model. Doubling is caused by the CAVE's stereoscopic display.

perform, and unambiguous enough that it was never performed accidentally, although some false positives still occurred due to tracking errors when the user was turned away from the Kinect.

Furthermore, the system-controlled procedure of holding a hand still for two seconds in order to release a grabpoint was reported as being unpleasant to use. To remedy this, I used the same change that I had used in releasing the axis cross in the zoom technique. A user could pick up a sphere in one hand, and then release the sphere by bringing the other hand close. This required that the other hand was not itself engaged in moving a grabpoint. I therefore specified that only one grabpoint could be picked up at a time. Like with the axis cross from the zoom technique, a grabpoint could not be picked up again for one second after it was placed.

Informal evaluation by colleagues revealed that the view manipulation procedure had many steps that could reasonably be shortened or omitted, making the entire interaction technique easier to use. To that end, some improvements were made: if the posture used to summon the interface was maintained for an additional two seconds, both grabpoints were locked where they were, and the view manipulation mode was immediately engaged. This allowed the user to begin view manipulation in a single step if the grabpoints were placed well enough. Informal experimentation showed that the initial placement of the grabpoints was not critical for good results, as the user could compensate for inaccurate placement during the actual view manipulation. Therefore, in many cases, the exact placement of the grabpoints was unimportant, and the view manipulation could be engaged in one step by maintaining the initial posture for three seconds.

Another improvement was to lock a grabpoint after it was manually placed once, since it was observed that grabpoints were never moved twice. This way, once both grabpoints were manually placed, they were also locked without the user needing to assume the initial posture again. These improvements helped to turn a somewhat cumbersome interaction technique in to one that could be learned and used with relative ease.

3.9 SYMBOLIC SYSTEM CONTROL: ADAPTED 2D MENU

Once the view manipulation techniques were in place, allowing users to explore a data visualization, I turned my attention to supporting users in manipulating the visualization itself. The types of symbolic manipulations to be supported, and the number of parameters for each manipulation, were variable. Therefore, creating a new interface for each manipulation, including a posture trigger, was not feasible, as this would make the learning curve for the interface far too steep. I chose a menu as an extensible interface technique that would support any number of manipulations.

I considered a number of menu techniques: a traditional 2D menu placed within the 3D environment, natively 3D menu widgets such as the Command and Control Cube (Grosjean et al., 2002), and 1 Degree-of-Freedom menu systems such as the Ring Menu (Liang and Green, 1994). I also considered a user-oriented menu system, which could take advantage of proprioception as explained in (Mine et al., 1997). In such menus, menu items are place over particular parts of the user's body, and the user can activate them by pointing to the body part. For example, a user might point with the left hand to the right elbow to activate the zooming mode. However, I found that Kinect tracking suffered when a user pointed towards his or her own body, making a user-oriented menu system unsuitable. I decided to begin with a prototype using a 2D menu, adapted to the 3D environment. This later turned out to be quite successful, and intuitive to use, and no other techniques were implemented.

The 2D menu consisted of a number of text labels. Such labels were already a part of the 'osgrc' application. Based on the OSG Billboard, these text labels were made to orient themselves to the virtual camera, so that the user always saw them from the front, even if the user moved relative to the labels. These text labels were, however, difficult to see, especially in the midst of a virtual model. To increase visibility, I placed the text labels against a colored background.

A number of these text labels were placed in a column. This column was placed somewhat in front of the user and somewhat to the right (somewhat to the left for left-handed users). The idea behind this was that placing the menu right in front of the user might obscure his or her working area. Furthermore, placing the menu to the right discouraged the user from going across with the left hand to select menu items, which would be likely to result in tracking errors.

Initially, the menu's location was continually updated so that if the user moved, the menu moved with him. Informal evaluation showed this to be unnecessarily complicated. I corrected this problem by

determining the menu placement once when the menu was invoked, and using that location until the menu was dismissed again.

Selecting a menu item was done by touching it. I initially did a simple intersection test using the location of the hand, with a margin of an additional 10 centimeters. I found that this intersection test did not perform as well as expected, as the hand joint was most prone to tracking errors. Using the wrist joint alleviated this problem, but did change the interaction so that now users had to somewhat 'push through' menu items to select them, rather than simply touch them. This could be prevented by extending the vector from elbow to wrist to create a more stable 'virtual' hand joint location, but during testing I found that users quickly became used to the way the menu operated.

A bigger problem was that, due to inaccuracy of the tracking data, users often selected the wrong menu item by mistake. This might be corrected by narrowing the margin in detecting intersections between the joint and a menu item, however this would also make it more difficult for the user to select the right menu item. Another solution would be to space the menu items further, but space in the CAVE was limited, especially since all menu items needed to be within physical reach of the user. In the end, this problem remained unsolved.

Since the menu needed to support a number of operations, some of which required multiple steps, sub-menus were needed. I considered adding sub-menus in the traditional 2D way, that is by placing them next to the menu item that was used to invoke them. However, I concluded that given the limited space available in the CAVE, and the relatively large menus, space would quickly run out. To solve this problem, I decided to make use of the third dimension, unused by the 2D menu. I made it so that when a menu item was selected that invoked a sub-menu, the current menu as well as any previously invoked menus would collectively be moved away from the user, and the new sub-menu be placed where its parent menu used to be. Since I anticipated that an instant change would likely confuse the user, I added in an animation where the previous menus moved away from the user in a one second animation, before the new sub-menu was displayed.

The use of the third dimension turned out to be quite intuitive: by making menus slightly transparent, the user could see through the current sub-menu to the parent menu, and its parents all the way to the first menu invoked. By changing the color of selected menu items, this created a sort of 'breadcrumb trail', where the user could track his or her path through the menu that had led to the current sub-menu. In each sub-menu, a 'back' option was given. When this was selected, the sub-menu disappeared, and the previous menus moved back forward in a one second animation. An example of this is shown in figure 16.

The menu is first invoked by clapping both hands together, then moving them apart again. I selected this gesture as it is quite simple, natural to make and easy to remember. Since using the menu was expected to be one of the most common operations, the invocation gesture needed to be simple as well. The menu could be dismissed again at any time by clapping again.

Unfortunately, I had great difficulty in preventing false positives in the detection of clapping. The clap detection algorithm was quite

Figure 16: A user using the menu interface. The user has selected the 'layers' option in the main menu, and that menu item has now been colored green, while other main menu items are displayed in red. The main menu has been moved back, and the 'layers' menu has been placed in front of it. The menu item for the 'skin' layer is red, indicating that this layer is currently not being displayed, while all other layers are being displayed. In the right bottom of the menu, menu items allow the user to simultaneously disable or enable all layers. A yellow menu item marked 'back' allows the user to return to the main menu. CAVE stereoscopic display was disabled for this picture.

simple at first, checking only for a moment where the two hand joints were within a predefined distance of each other in one cycle of the program, and outside this range in the next. After witnessing a great number of false positives, causing the menu to appear when this was not desired, I tried a number of methods to make the detection more robust.

I added a check whether the position of joints relative to each other in the skeleton did not show any obvious deviations, to prevent clap detection from faulty data due to tracking errors. Unfortunately, most tracking errors left the skeleton mostly intact, only being incorrect in the location of one joint that was obscured to the Kinect.

I also checked the user's orientation, only allowing claps to be detected when the user was facing forward to within a certain margin. False positives still occurred when one arm was obscured to the Kinect. Using the exoskeleton, I found that the obscured arm was placed directly beside the visible arm. To counter false positives in this situation, I added a check whether the arms were a certain distance apart during clapping, even if the hands were close together. Taken together, these measures greatly reduced the number of false positives in clap detection, but did not eliminate it completely.

The basic functioning of the menu did not change further. During evaluation, some changes were made to one particular sub-menu, which will be described in section 4.7.1.

3.10 LIMITATIONS

These were all the interface features I was able to implement during this project. Before I turn to the evaluation of the system, I will discuss some of the drawbacks of this prototype that were known before evaluation began.

While the Kinect can track up to six people within its field of vision, only two of these can be 'actively tracked', i.e. have feature extraction applied so that the location of their joints in 3D space is known. On the assumption that the prototype could only support one user at once, the Kinect data server I built only provides skeleton data for a single user. However, the Kinect was oriented toward the entrance of the CAVE, and people walking across from the entrance outside the CAVE would enter the Kinect's field of vision. One persistent problem was that such people would start to be actively tracked, while active tracking of the current interface user was lost. I tried to correct this problem in the Kinect server by only sending the tracking data for the user closest to the Kinect, however for unknown reasons this did not solve the problem, and time constraints prevented me from working on the problem further. As a workaround, I asked spectators to keep some distance so as not to enter the Kinect's field of vision, but for a production-level interface, this problem needs to be resolved.

Other than this, the main problem for the prototype was one of accuracy. Two factors adversely affected accuracy: tracking errors and mapping errors. Tracking errors occurred when parts of the user's body were (partly) occluded to the Kinect camera. This could happen even when hands were merely held in close proximity to each other, causing the estimated position of either hand to vary wildly. Mapping errors occurred mostly in the CAVE areas furthest left and right: in these locations, estimated hand position could be as much as a few decimeters off from the actual position.

These inaccuracies could adversely affect the use of the interface. Particularly, it could lead to false positives and misses in the detection of postures and gestures, as well cause sudden changes or jitter when the user was using his or her hands to input parameters to a command. The view manipulation technique, for example, tended to shake the entire model when the user crossed their hands or held them in close proximity to each other.

These problems could most likely be solved by the application of signal analysis techniques. If false data from tracking errors could be ignored, or smoothed, the functioning of the system would improve markedly. However, the signal quality was good enough that evaluation could proceed regardless. It proved entirely possible to complete the commands offered by the interface without encountering any errors.

In the next chapter, I will describe how the prototype system was evaluated by having test participants work with each of the interfaces. It will be described there that, although the users did always encounter tracking errors at least once, these were not so prevalent as to disrupt their use of the interface.

4 EVALUATION

In previous chapters I have described how, in an effort to support prop-free, casual 3D interaction with virtual environments, I developed a prototype system using the Kinect as a tracking input device, and the CAVE as a virtual reality output device. This prototype system included two games, Pong and Bubble Burst, as well as interfaces for object manipulation and data exploration, supporting both view manipulation and symbolic manipulation of the model through a menu.

In this chapter I describe an informal evaluation held to consider the functioning of the prototype as it was used by novices. To that end, test participants were invited to use the prototype in a number of tasks, and then fill out a questionnaire. In addition to their answers, I have also observed their use of the prototype, and the nature and origin of errors that were made. In this chapter, I describe the evaluation, the questionnaire results, the feedback I received from test participants and the errors I observed.

4.1 RATIONALE

The previous chapter describes the development of the prototype system. During primary development, I tested new parts of the interface myself, or asked colleagues to test them, in an effort to find and fix potential usability problems.

However, I and most of my colleagues were technical experts and quickly grew used to using the prototype system. This prevented us from detecting problems with the system that a new user might experience. Furthermore, we needed to determine how well the system performed on such criteria as learnability, usability, comfort, fun, being conducive to a feeling of presence and, most importantly, available for casual use. This required new users, untrained in the use of the system, to evaluate the current state of the system.

To that end, an informal evaluation was organized which would nevertheless be stricter and give more meaningful results than the informal testing that had been performed as part of the design process.

4.2 EVALUATION SETUP: DISSECTING A FROG

Since the change in application domain (see section 3.7), the system had been geared toward data exploration. The system had been developed and tested on an anatomical model of a frog. This model consisted of automatically segmented scan data, and contained a number of separate objects such as a heart, stomach and skeleton. To test the system using the frog model, we wanted to invite biology students.

An announcement was placed on the digital notice board for the biology department at the Rijksuniversiteit Groningen. To encourage more participants to come to the evaluation, we also accepted students coming in groups of two or more. Eleven students accepted the invitation. Six students came in groups of two, one group consisted of three students and two students came individually. The average age of the students was 20 years, five were female and six male. Each study session took about an hour to an hour and a half.

The studies were held in a somewhat informal setting. Tasks were very loosely defined, and no performance metrics were measured, leaving room for students to explore the use of the interface on their own at ease. When multiple participants were present in a study, one participants would make use of an interface while the rest watched from a distance. Observing how one participant could take another's place in the CAVE and learn from what they had seen their predecessor do turned out to be very informative: see section 4.5.

Each study would begin with a short demonstration of the capabilities of the CAVE. This was followed with an automated tutorial to the capabilities of the Kinect, explaining how the prototype system was set up and what kind of tracking errors the user could expect. This prepared the participants to the use of the prototype, and allowed them to anticipate and correct for inaccuracies due to tracking errors. However, the first evaluations showed that users did not enjoy the use of this tutorial, and it also consumed more time then when the experimenter explained the inaccuracies personally. Therefore, the tutorial was skipped with the last group of two participants. During the tutorial, the exoskeleton would be displayed to familiarize the user with Kinect tracking and its shortcomings. When the tutorial was skipped, the exoskeleton was still used while the experimenter explained the inaccuracies of the system.

After this, the evaluation began with a game of Pong. This familiarized users with how tracking and mapping errors affected their use of the interface, as well as make them enthusiastic for the potential of the interface. After Pong, the participants played several rounds of Bubble Burst, at least three rounds per participants per selection mechanism, so that participants could reliably compare all three selection mechanisms.

Finally, the frog model was loaded. The experimenter demonstrated both view manipulation techniques, zooming and free view manipulation, as well as the use of the menu. After this, participants were left free to explore the model and the interface on their own. Occasionally, participants might be asked to perform a certain task, such as disabling or enabling a particular layer, taking a particular viewpoint, or naming a particular organ in the model. If a group of multiple participants was present, the interaction techniques were demonstrated once, and participants took turns in applying them to the model.

4.3 QUESTIONNAIRE SETUP

After the participants had used each part of the prototype, they were asked to fill out an evaluation form. This form consisted mostly of 5-point Likert scale questions, assessing whether the participants agreed

with a particular statement. For each interaction technique, where applicable, statements were presented that touched on whether that technique

- was conducive to a sense of presence
- was comfortable in use
- was accurate in following the user's motions
- appeared / started without being called for
- was easy to learn
- was easy to use once learned
- behaved in unexpected ways
- was fun to use

In addition, participants were asked to compare and rate the different selection mechanisms they had encountered during Bubble Burst. Three questions asked the participant to choose the selection technique that was

- most reliable
- easiest to use
- overall best

The final question of these three also offered room for the participant to note the reasons why this technique was rated best.

Finally, participants were asked to note their age and rate their familiarity with computer interaction on a 5-point Likert scale from unfamiliar to expert.

There was no explicit room for free commenting on the evaluation form, but participants were encouraged to share their views with the experimenter. This feedback was either used to immediately improve the system, as described in section 4.7, or was noted down for discussion in section 4.6.

4.4 QUESTIONNAIRE RESULTS

The questionnaire had a noticeable positive bias: participants tended to rate the the comfort, learnability, usability, etc. of all techniques as positive. If 'completely disagree' were given a rating of -2, and 'completely agree' a rating of 2, then the average response from participants on all questions was 0.67, with a standard deviation of 0.91. This may be due to a confirmation bias, or to the fact that participants were somewhat overwhelmed by the possibilities of the prototype, since most participants stated that they had no experience with immersive virtual reality. More rigorous testing, e.g. using control groups or participants more familiar with IVE systems, may compensate for this bias.

Of the three selection mechanisms explored during the Bubble Burst game, participants rated 'touch' as easy to use, while 'hold' and 'enclose' were considered more reliable, as can be seen in figure 17. The mechanism rated as best varied with participants: some preferred the simpler 'touch', others the more reliable 'hold' and 'enclose'.

It was found that the Bubble Burst game in particular increased the feeling of presence, with Pong and the view manipulation techniques scoring somewhat lower, and the symbolic menu, as well as the more passive exoskeleton and tutorial, rating lowest (see figure 18a).

Participants rated the Bubble Burst game as the most comfortable in use, followed by the tutorial and the free view manipulation technique. The zooming technique was rated as particularly uncomfortable in use (see figure 18b).

The Bubble Burst game was also rated as being accurate in tracking the participant's movements, while the Pong game was rated as being inaccurate, as shown in figure 19a.

Figure 19b shows that the free view manipulation technique was rated as seldom appearing unbidden, unlike the menu and zoom techniques.

Most interaction techniques were found easy to learn, as shown in figure 20a, with the Bubble Burst game and the menu in particular.

Bubble Burst and the menu were also found easy to use, as was the free view manipulation technique, with the Pong game and zoom technique rated less so (see figure 20b).

Participants found that the menu often acted in unexpected ways, as seen in figure 21a. The zoom technique was found to act more as expected, free view manipulation less so.

Figure 21b shows that most interaction technique were considered fun to use, with the exception of the automated tutorial, and the exoskeleton and menu rated less fun than the other techniques. The Bubble Burst game in particularly was almost unanimously agreed to be fun to use.

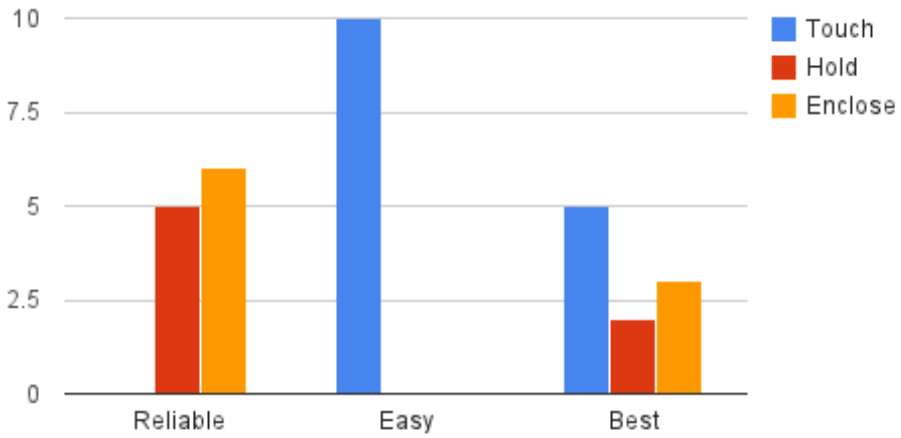
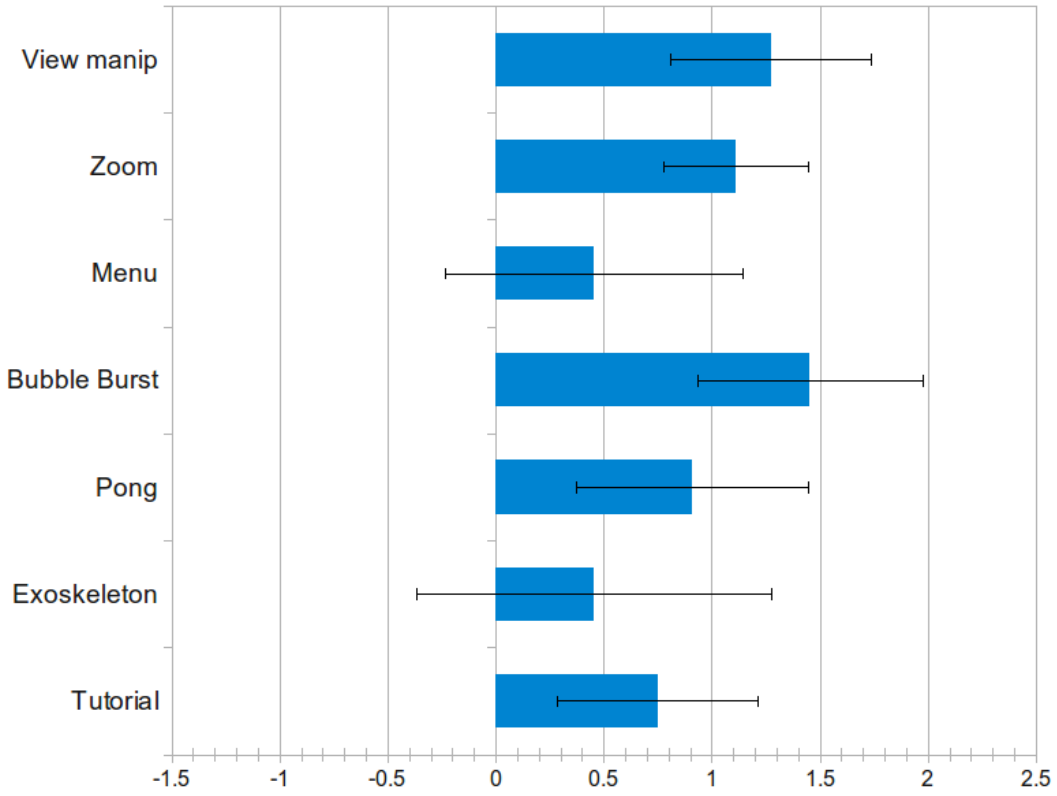
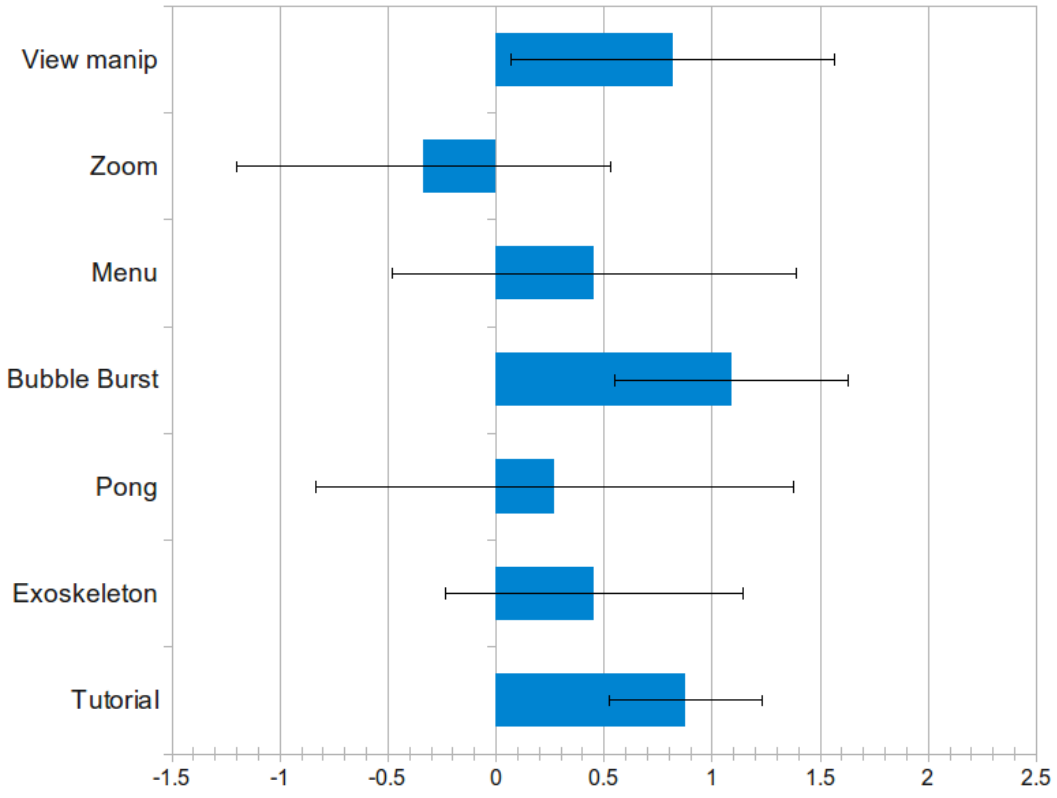


Figure 17: Number of times each selection mechanism was rated highest according to the given criterion.

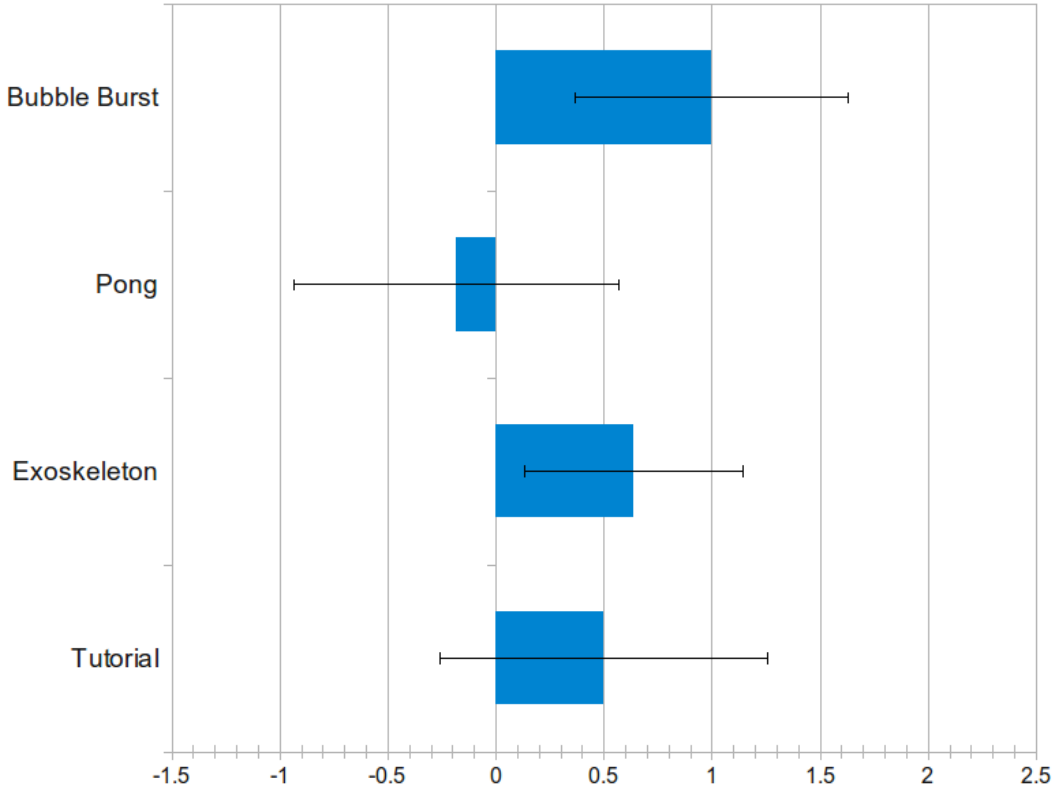


(a) Technique X enhanced my sense of presence in the CAVE.

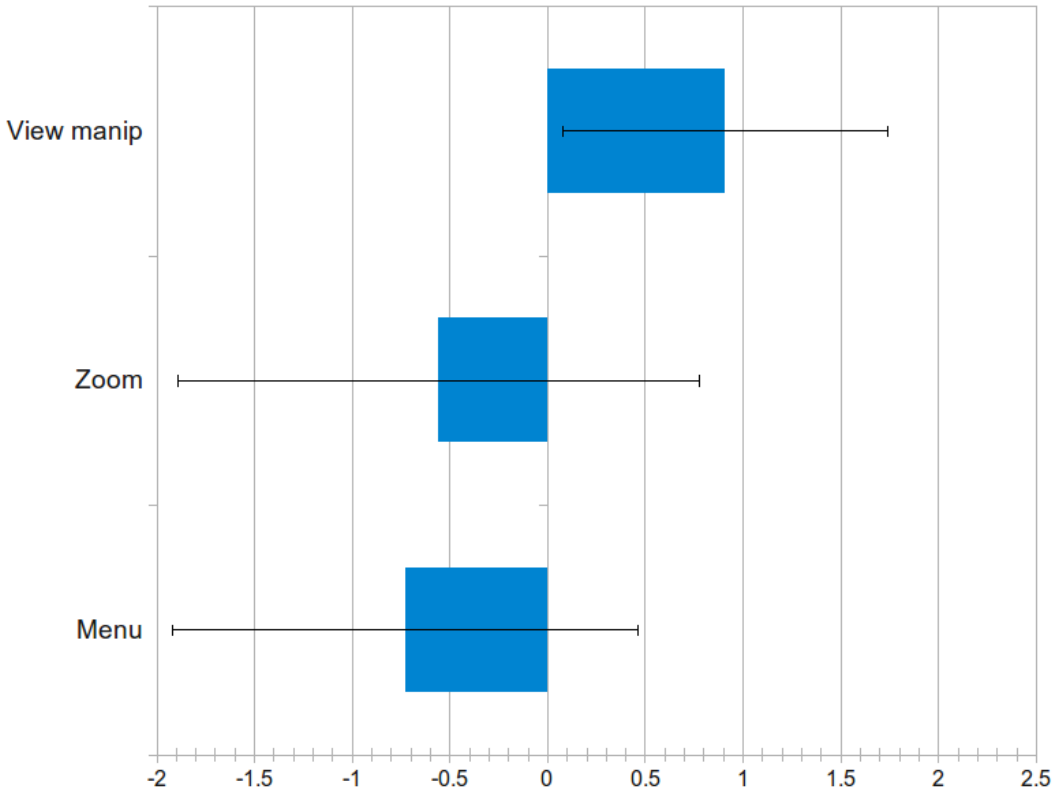


(b) Technique X was comfortable in its use.

Figure 18: Average participants rating of agreement with statements testing sense of presence and comfort on a 5 point Likert scale, according to interaction technique, where 0 is neutral, 2 is 'completely agree', and -2 is 'completely disagree'. Error bars represent one standard deviation.

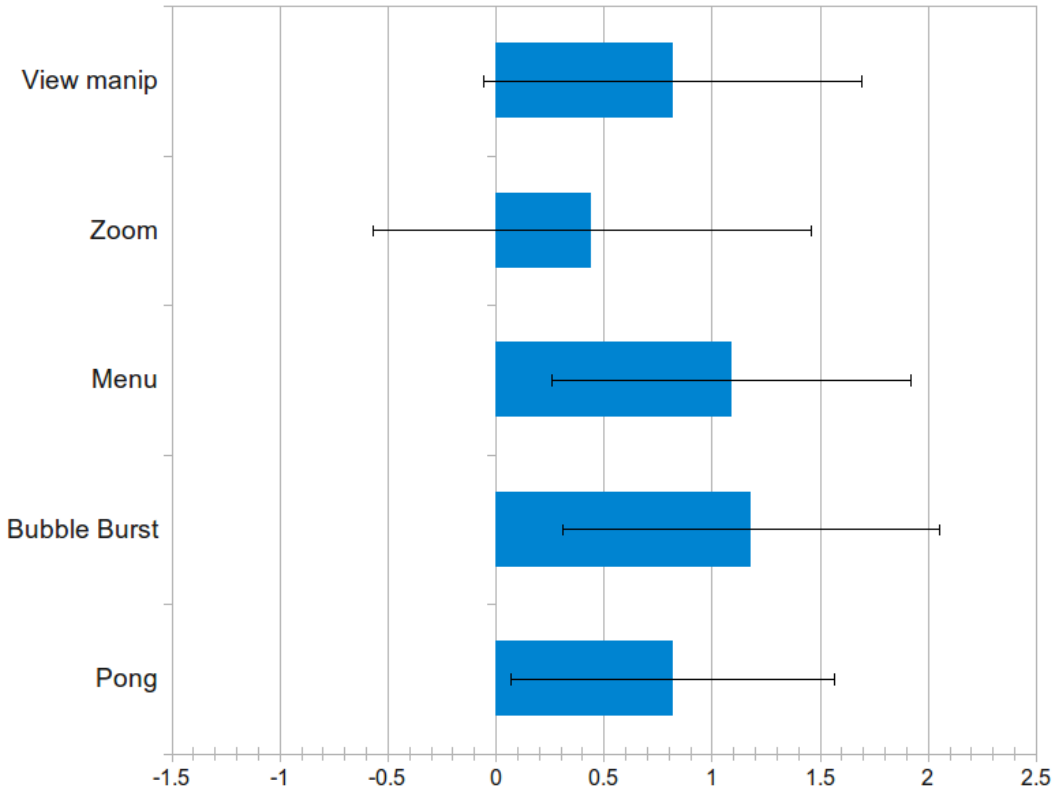


(a) Technique X was accurate in following my movements.

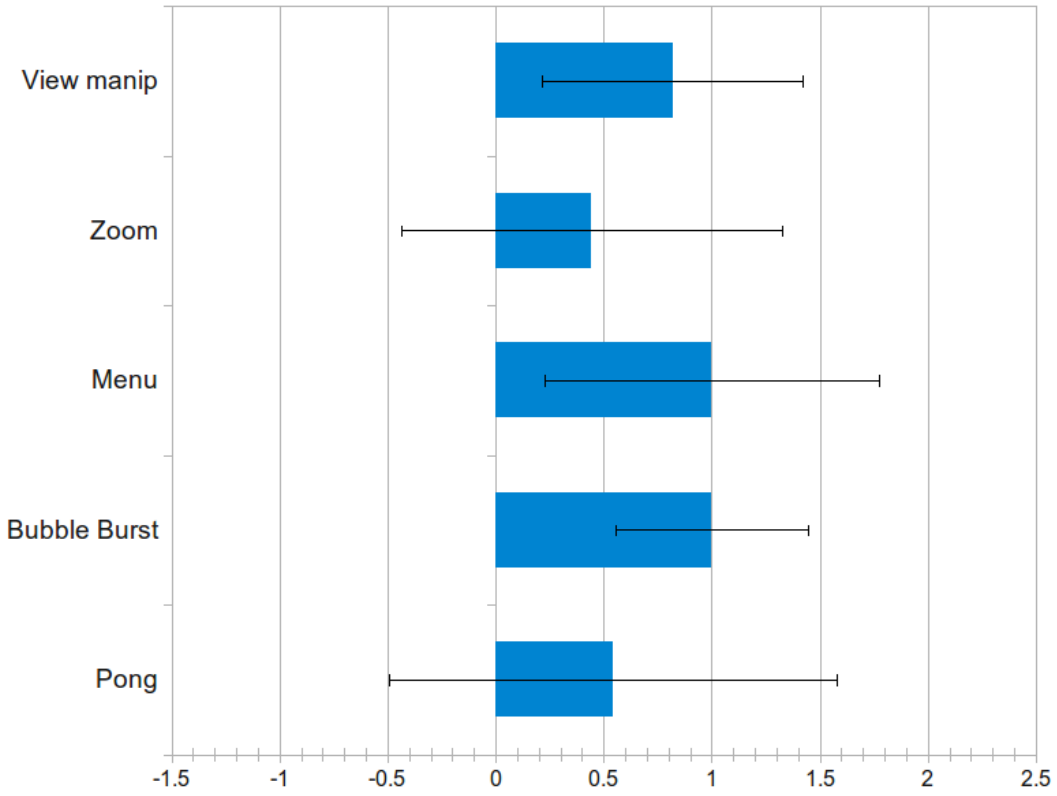


(b) Technique X never appeared uncalled.
Note that a different scale is used from other graphs.

Figure 19: Average participants rating of agreement with statements testing accuracy and whether a technique ever appeared uncalled, rated on a 5 point Likert scale, according to interaction technique, where 0 is neutral, 2 is 'completely agree', and -2 is 'completely disagree'. Error bars represent one standard deviation.

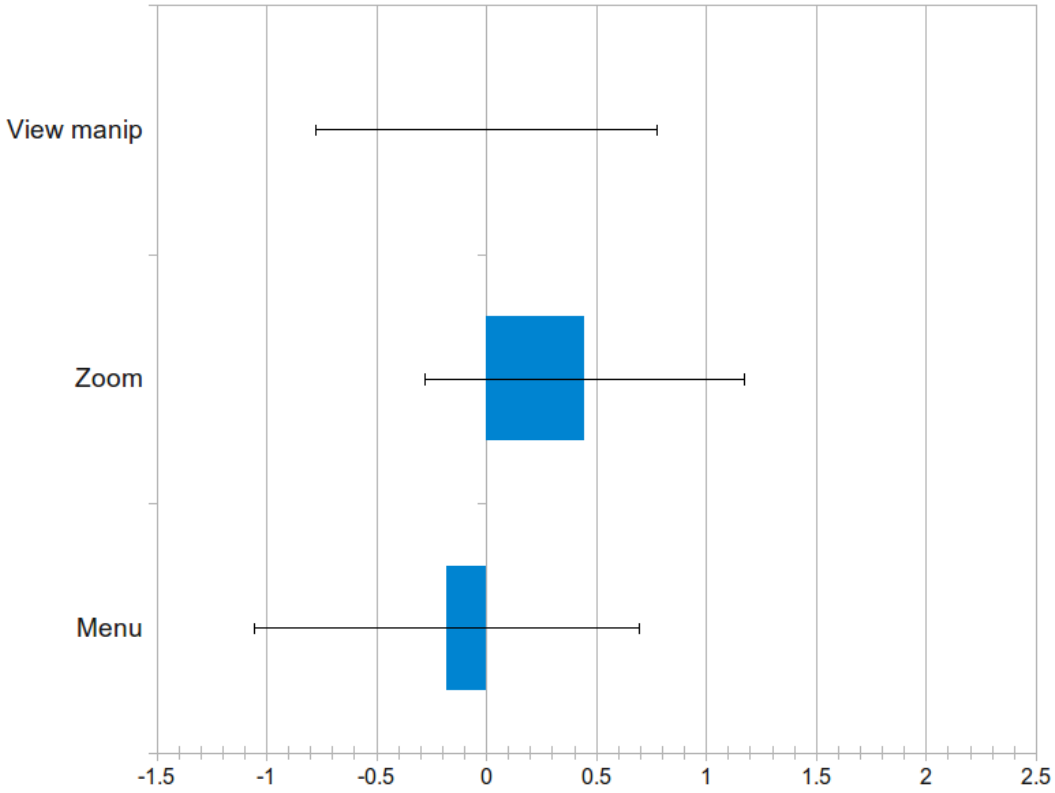


(a) Technique X was easy to learn.

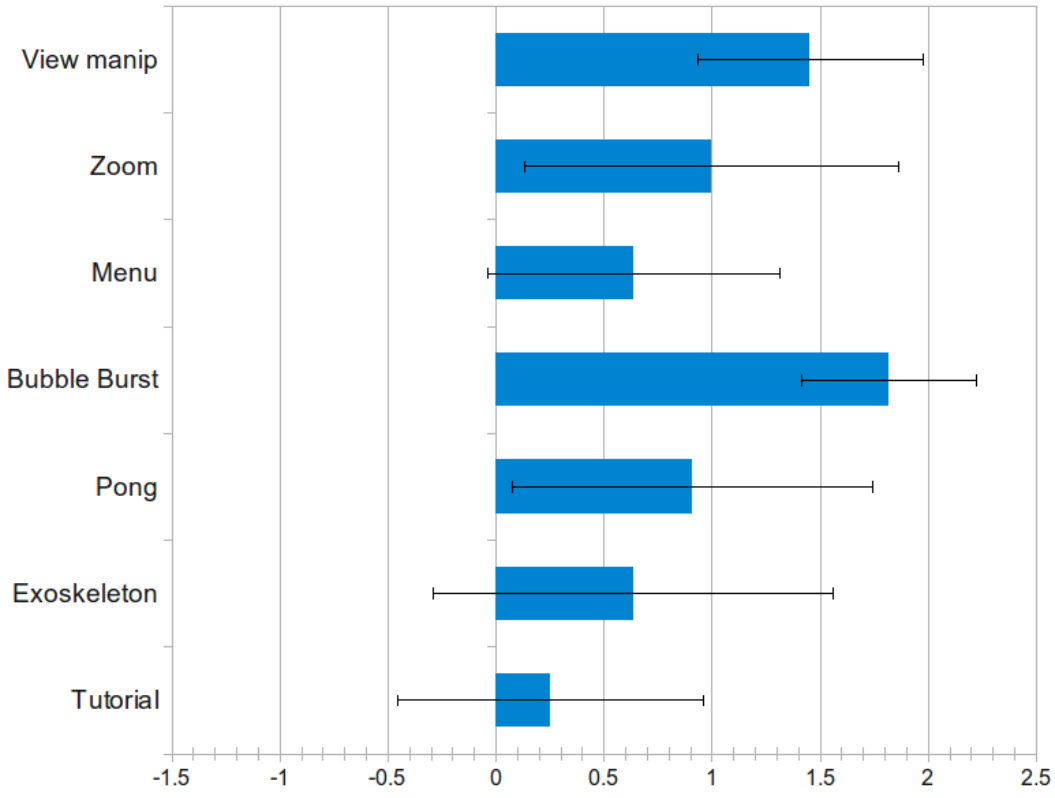


(b) Once learned, technique X was easy to use.

Figure 20: Average participants rating of agreement with statement testing learnability and usability on a 5 point Likert scale, according to interaction technique, where 0 is neutral, 2 is 'completely agree', and -2 is 'completely disagree'. Error bars represent one standard deviation.



(a) Technique X never behaved in unexpected ways.



(b) Technique X was fun to use.

Figure 21: Average participant rating of agreement with statement testing whether a technique ever behaved in unexpected ways and was fun to use, rated on a 5 point Likert scale, according to interaction technique, where 0 is neutral, 2 is 'completely agree', and -2 is 'completely disagree'. Error bars represent one standard deviation.

4.5 CASUAL USE

The prototype was found to be quite effective as a means of supporting casual use of the 3D interface. In fact, when groups of multiple participants were present, the participants often had to switch a number of times during the experiment. In a few cases, this caused some confusion as the previous user was still actively tracked, due to the problem described in section 3.10. However, aside from this problem, users could change places with little difficulty: a new user could walk into the CAVE and immediately start using the interface.

Participants who observed another user being instructed in the use of an interaction technique, or even simply observed another user making use of an interface, usually did not need additional instructions: knowledge gathered from observation transferred quite easily to knowing how to use the interface themselves.

4.6 INFORMAL FEEDBACK

Feedback given verbally by the participants during the evaluation generally covered three topics.

Many participants complained that tracking errors often interrupted their use of the interface, and that this was the foremost problem they had with the prototype. These problems were partly anticipated (see section 3.10). The most important next step in developing this prototype would be to increase precision and prevent tracking errors, as discussed in section 5.3.

Other feedback concerned problems with distinct features of the interface, such as the inability to successfully engage the zooming mode, or difficulty understanding or using the layer sub-menu. These comments were used to correct such usability problems, as is discussed in section 4.7.

Most of all, participants were enthusiastic about the possibilities of the prototype. The Bubble Burst game in particular was praised as a fun game. And although most participants agreed that the exploration of the frog model was no replacement for the practical experience gained by dissecting a real frog, they also thought that the exploration of the frog model would make a good addition to the biology department's educational program.

4.7 PROTOTYPE ADJUSTMENTS

During evaluation, feedback from the participants was used to improve the system. Seven participants used the initial system, while the final four participants used a version with improved layers sub-menu (explained below) and the final two participants tested a version where the posture for summoning the zoom interface had been disabled (explained below).

4.7.1 *The Menu*

One of the sub-menus could be used to enable or disable layers of the simulation. A layer consisted of a number of virtual objects. In the

frog model model, for example, one layer consisted of all the virtual objects that together formed the frog's circulatory system.

During the evaluation it became evident that the use of this sub-menu was not at all intuitive. In the initial design, all layers were listed in a single sub-menu using a number of columns of menu items standing side by side in the CAVE. When one layer was selected, a new sub-menu appeared, offering the options of enabling this layer, disabling it, or enabling it while disabling all others. Users often accidentally selected this last option, when they had only intended to enable the layer, causing disruption to their intended task. Also, users became confused as to which layers were currently active, as no feedback in the sub-menu showed this.

On the advice of some of the first study participants, I redesigned the layer sub-menu. In the new design, layers were still displayed in a number of columns, but now the layers that were enabled had a green background, while the disabled layers were red. Furthermore, when one of the layer items was selected, the layer it represented was immediately toggled, without the need for an additional sub-menu. Finally, two options were added to the end of the list of layers, offering the option of simultaneously enabling or disabling all layers. Feedback from the next participants in the evaluation showed this new design, which is displayed in figure 16, to be far easier and more intuitive to use.

4.7.2 *Zooming*

During evaluation it became apparent that some participants had trouble engaging the zoom mode once the zoom point was placed (turned red): either because of tracking errors, or problems in hand-eye coordination, users were unable to touch both ends of the white bar at once. If the spheres were not touched within six seconds, the view manipulation was canceled when this was not the user's intention. This problem was simply solved by engaging the zoom mode once either end of the slider was touched, rather than both. Also, the 'cancel' posture replaced the timing that automatically canceled the interaction if it wasn't engaged.

It was described in section 3.8.1 that the posture used to summon the zoom interface was set to one hand held to the user's head, while the other hand was held over the point to zoom in on. However, during the user evaluation, this posture was often accidentally assumed by users as they were adjusting the reflector cap, or their 3D glasses. Each time, the zoom interface would appear unintended, causing great irritation in the participants.

As the zoom technique was seldom used by participants during the evaluation, and more use was made of the free view manipulation technique, I decided to forgo the use of a posture to trigger the zoom technique. The zoom technique remained available, but could only be reached through the menu. The final two test participants tested the version of the prototype where zooming had no dedicated posture, and it was observed that these participants did not use normal zooming at all, not even through the menu.

5 DISCUSSION

In an effort to create an interface that supports casual 3D interaction, I have designed a prototype using the tracking data provided by the Microsoft Kinect, coupled with the immersive virtual environment display capabilities of the CAVE. The prototype included a visualization of tracking data called the 'exoskeleton', two games, an interface for object manipulation and an interface for data exploration, including two view manipulation techniques and a menu-based symbolic manipulation technique. I have evaluated the design of this prototype by inviting biology students to use the system to explore an anatomical model of a frog, as was described in the previous chapter.

In this chapter, I will discuss the outcome of the evaluation, and consider whether the goal of a 3D interface for casual use has been attained. Finally, I will make suggestions about how research on this topic might process further, and whether the prototype is currently viable for productive use.

5.1 EVALUATION RESULTS

One of the most highly praised features of the prototype was the Bubble Burst game. People learned how to play the game quickly, could easily and comfortably use it, and reported that it tracked their movements precisely. Above all, the game was considered fun to play and increased the feeling of presence in the CAVE.

I attribute the high praise that Bubble Burst received to a couple of factors:

- The Bubble Burst game was relatively fault tolerant. Participants needed to select spheres that went up to quite large size, so that tracking errors did not prevent users from selecting such spheres. This may account for why Bubble Burst was considered easy and comfortable to use, and precise in tracking user movements, even though the input quality was no better than with other, more fault sensitive parts of the prototype.

- The interface of the Bubble Burst game was extremely simple. Subject were only required to touch or envelop the spheres, both quite natural gestures. No additional steps were required. The greatest complexity of the game from the user's perspective, namely considering which sphere was next in the chosen order, was part of the game, and seen as a challenge rather than a difficulty. This easy interface probably contributed to the game's learnability, fun, and the increased feeling of presence it caused in participants.

The results of the evaluation of the different selection techniques as used by Bubble Burst were to be expected to some degree. Techniques that separated indication from selection, namely 'hold' and 'enclose', and which had visual feedback for each phase, were considered more

reliable, while the 'touch' mechanism, which combined both steps, was considered easier to use. Participants differed of opinion which technique was better, probably because no further context was given with this question, and participants had different criteria to evaluate which technique was 'best'. Some opted for reliability, others for ease, and others for some combination of both.

The Pong game on the other hand was quite sensitive to tracking errors, causing users to miss hitting the ball when they were sure they had reached for it correctly. This is probably why Pong was evaluated to be far less comfortable or easy to use as Bubble Burst and other interaction techniques. It was also considered to be far more imprecise than other interaction techniques, not because the quality of the signal was different, but because the consequences of tracking errors became far more obvious to users. However, the game was considered to add to the feeling of presence in the CAVE. This may be due to the simplicity of the game adding to a feeling of transparency.

The zoom technique was considered by many to be hard to learn and use, although it did function more predictably than free view manipulation. It was also observed that the zoom technique was used by most participants only once or twice, while the free view manipulation technique was used far more often. In fact, during the last study, when the trigger posture for zooming was removed, the zoom technique was not accessed at all. Moreover, when the trigger posture was still present, it was quite often detected accidentally, to the irritation of users. As part of the data exploration task, the zooming technique did increase the feeling of presence, probably because it immersed participants in the model itself when zooming in. But although zooming does allow a greater range in zooming in and out, the additional freedom and more intuitive use of free view manipulation caused most participants to prefer it over zooming.

In fact, of all the interaction techniques in the data exploration task, including zooming and the menu, free view manipulation was used most often, and was evaluated best. It was considered reasonably comfortable and easy to use. It increased feelings of presence, and was considered fun to use. However, parts of the technique of free view manipulation were considered unexpected, and a bit harder to learn than other techniques. Possibly this was due to the extra steps needed to engage free view manipulation (which were removed at one point after the evaluation) or because tracking errors sometimes caused the entire model to jump location or orientation unexpectedly. The trigger posture for free view manipulation also proved unambiguous enough that it was almost never detected accidentally.

In contrast, the menu appeared accidentally most often. This was due to the simplicity of the gesture required to invoke it, namely simply clapping hands together. However, the simplicity of this gesture may also have contributed to the menu being considered quite learnable and easy to use. The action of pressing menu items to select them were considered quite natural, and were often tried without prompting by the experimenter. Unfortunately, inaccuracies in the tracking data often caused the incorrect menu item to be selected, which is why the menu was reported to often behave in unexpected ways, and evaluated not quite as comfortable to use as some of the other

interaction techniques. The menu was not considered as conducive to a feeling of presence in the CAVE as other techniques, possibly due to the fact that it disrupted the user's workflow or because of the unexpected behavior it displayed.

5.2 CONCLUSION

My goal has been to create a prototype that would support casual 3D interaction, and in this I have been successful. Especially considering the relatively small amount of time invested to develop the prototype, it performed quite well during the evaluation. Participants learned how to use the various interaction techniques with ease, and although tracking errors often disrupted their work, they still found it quite easy to work with the interface.

Some usability problems persisted into the version of the prototype used with the first groups of participants. However, even during the evaluation I continued to improve the interface based on feedback, by the time the last group of participants arrived, most of these problems had been corrected. Some lessons learned from the design and subsequent improvement of this prototype may be of help to future designers of 3D interfaces.

- The less steps are needed to engage an interface, the easier it will become to learn and use. Fewer steps will make the interface more transparent, increasing the feeling of presence for users of the interface.
- Errors that occur should be made easy to correct. For example, introducing a posture to cancel an interaction that was invoked accidentally greatly increased the usability of this prototype.
- Behavior of the interface that is driven by the system should be avoided. By giving users control over when an interaction begins and ends, such interactions become far easier to use. When the user controls the interaction, rather than the system, this also allows users to create mental shortcuts or motor programs to quickly execute actions using that interaction technique.
  Unfortunately, there remained a number of occasions in my prototype where full user control proved infeasible, particularly where both the user's hands are occupied inputting command parameters. The only way to prevent these system-controlled changes of mode would be to find other ways for the user to input these parameters, or find ways for the user to indicate mode changes without moving his or her hands.

5.3 FUTURE WORK

As has been mentioned time and again throughout this report, the greatest drawback of this prototype is its inaccuracy and tendency toward tracking errors. Tracking errors occur most often because of joints being obscured to the Kinect, causing the Kinect to guess in its extrapolation of the obscured joint's location. A short-term but simple solution would be to perform some kind of signal analysis

that would allow the interface to ignore such faulty input. When the interface is being used, most joints involved in the interaction are usually unobscured, so that ignoring obscured joints will not necessarily degrade the functioning of the system.

A better solution, but one that would take more time and effort to implement, is to add more Kinect sensors. With overlapping fields of vision, it could be made so that joints are seldom or never obscured to all Kinect sensors. This would still require that the data from all sensors be correlated. If two sensors report a different location for a user's joint, only the sensor that has a better viewing angle should be relied on. Fortunately, efforts on this topic are already underway (e.g., Sumar and Bainbridge-Smith, 2011; Schröder et al., 2011).

Mapping errors can be reduced simply by manually improving the coordinate mapping from Kinect to CAVE, but it might be even better to develop a more exact calibration method, since even the most sensitively tuned mapping will become inaccurate if the position of the Kinect changes even slightly in relation to that of the CAVE.

Once the accuracy and reliability of the system are increased, one might look into increasing its functionality. Although the current prototype has shown that it is possible to solve the User Intention Problem by the use of posture and gesture triggers, it would be far more intuitive for users to add support for detecting discrete events to use as triggers. The most obvious example, as suggested in section 3.1.2, would be to detect from the Kinect depth sensor data, RGB camera data, or both, whether the user has his or hands open, or closed. This simple addition could be used to make free view manipulation, which at the moment is still a bit complicated to initiate, a fully natural interaction: the user would simply physically grab the virtual world in two places, transform the view of the model, then release the world again to end the interaction.

Finally, the entry requirements for this prototype can be lowered by removing each of the remaining artifacts that are currently still required to use the system: a cap for head tracking and a set of shutter glasses. As explained in chapter 1, the former could be removed if the Kinect took over the function of head tracking, while removing the latter would be more difficult, requiring a form of autostereoscopy be introduced to the CAVE.

5.4 Viability for production level applications

The tracking errors that regularly occur in the prototype currently make it unsuited for production level applications. Although the prototype performed well enough in an informal evaluation on a data exploration task, an actual scientific visualization tool would require more reliability and precision than the system currently offers.

However, as was remarked in the previous section, it should be a relatively straightforward task to improve the robustness of the system. If the frequent tracking errors can be removed, the system becomes suitable to a range of applications. This prototype has shown data exploration to be one such application, and education potentially another. The low demands that this system places on the user, especially if either or both remaining required props can be removed, will mean

that virtual worlds become something a user can simply walk into, as easily as any physical environment.

What this system is not suited for, and probably never will be, is any task that requires high degrees of precision, such as CAD or any kind of scientific visualization task that also requires exact measurements. In these cases, it is more practical to rely on 3D controllers that can offer the required accuracy. Besides, the major advantage of this system, that of affording casual interaction, is one that hardly applies to such involved and complex applications.

However, technology is always improving. The Kinect was one of the first affordable optical input devices that allowed for prop-free 3D tracking. Now that it has revealed the wide range of applications to which such hardware may be put, new, more precise devices are already being developed. Therefore, a time may come when optical prop-free tracking becomes the standard for computer input, and we will use controllers and props in our interfaces only where we have good reason to do so. It will be a world where the virtual world becomes as easy to interact with as the physical one, and the borders between the two will start to melt away.

Azuma, R. and Bishop, G. (1994). Improving static and dynamic registration in an optical see-through HMD. In *Proceedings of the 21st annual conference on computer graphics and interactive techniques*, pages 197–204. ACM.

Bellucci, A., Malizia, A., Diaz, P., and Aedo, I. (2010). Human-display interaction technology: Emerging remote interfaces for pervasive display environments. *Pervasive Computing, IEEE*, 9(2):72–76.

Bolt, R. (1980). "Put-that-there": voice and gesture at the graphics interface. In *ACM Siggraph Computer Graphics*, volume 14, pages 262–270. ACM.

Bowman, D. and Hodges, L. (1997). An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environments. In *Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 35–ff. ACM.

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Brooks Jr, F. (1999). What's real about virtual reality? *IEEE Computer Graphics and Applications*, 19(6):16–27.

Bryson, S. (1996). Virtual reality in scientific visualization. *Communications of the ACM*, 39(5):62–71.

Burns, D. (1998). OpenSceneGraph. `http://www.openscenegraph.org/`.

Cruz-Neira, C., Leigh, J., Papka, M., Barnes, C., Cohen, S., Das, S., Engelmann, R., Hudson, R., Roy, T., Siegel, L., Vasilakis, C., DeFanti, T., and Sandin, D. (1993a). Scientists in wonderland: a report on visualization applications in the cave virtual reality environment. In *Proceedings of the IEEE 1993 Symposium on Research Frontiers in Virtual Reality*, pages 59 –66.

Cruz-Neira, C., Sandin, D., and DeFanti, T. (1993b). Surround-screen projection-based virtual reality: the design and implementation of the CAVE. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 135–142. ACM.

Cruz-Neira, C., Sandin, D. J., DeFanti, T. A., Kenyon, R. V., and Hart, J. C. (1992). The CAVE: audio visual experience automatic virtual environment. *Communications of the ACM*, 35(6):64–72.

Fisher, S., McGreevy, M., Humphries, J., and Robinett, W. (1987). Virtual environment display system. In *Proceedings of the 1986 workshop on Interactive 3D graphics*, pages 77–87. ACM.

Gallo, L., Placitelli, A., and Ciampi, M. (2011). Controller-free exploration of medical image data: experiencing the Kinect. In *24th*

International Symposium on Computer-Based Medical Systems (CBMS), pages 1–6. IEEE.

Grosjean, J., Burkhardt, J., Coquillart, S., and Richard, P. (2002). Evaluation of the command and control cube. In *Proceedings of the Fourth IEEE International Conference on Multimodal Interfaces (ICMI'02)*, pages 473–478. IEEE.

Keefe, D., Feliz, D., Moscovich, T., Laidlaw, D., and LaViola Jr, J. (2001). CavePainting: a fully immersive 3D artistic medium and interactive experience. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 85–93. ACM.

Keskin, C. (2006). *Vision based real-time continuous 3D hand gesture recognition interface for generic applications based on input-output hidden markov models*. PhD thesis, Bogaziçi University.

Lai, K., Konrad, J., and Ishwar, P. (2012). A gesture-driven computer interface using Kinect. *Image Analysis and Interpretation (SSIAI)*, pages 185–188.

LaViola, Jr., J. J. (2000). MSVT: a virtual reality-based multimodal scientific visualization tool. In *Proceedings of the Third IASTED International Conference on Computer Graphics and Imaging*, pages 1–7.

Liang, J. and Green, M. (1994). JDCAD: a highly interactive 3d modeling system. *Computer and graphics*, 18(4):499–506.

Lombard, M. and Ditton, T. (1997). At the heart of it all: The concept of presence. *Journal of Computer-Mediated Communication*, 3(2).

Loomis, J., Blascovich, J., and Beall, A. (1999). Immersive virtual environment technology as a basic research tool in psychology. *Behavior Research Methods*, 31(4):557–564.

Matikainen, P., Pillai, P., Mummert, L., Sukthankar, R., and Hebert, M. (2011). Prop-free pointing detection in dynamic cluttered environments. In *Automatic Face & Gesture Recognition and Workshops (FG 2011), 2011 IEEE International Conference on*, pages 374–381. IEEE.

Mine, M., Brooks Jr, F., and Sequin, C. (1997). Moving objects in space: exploiting proprioception in virtual-environment interaction. In *Proceedings of the 24th annual conference on Computer Graphics and Interactive Techniques*, pages 19–26. ACM Press/Addison-Wesley Publishing Co.

Norman, D. (1981). Categorization of action slips. *Psychological Review*, 88(1):1.

Oikonomidis, I., Kyriazis, N., and Argyros, A. (2011). Efficient model-based 3D tracking of hand articulations using Kinect. *Proceedings of BMVC, Dundee, UK*.

openNI (2011). Introducing OpenNI. `http://www.openni.org`.

Poupyrev, I., Billinghurst, M., Weghorst, S., and Ichikawa, T. (1996). The go-go interaction technique: non-linear mapping for direct manipulation in VR. In *Proceedings of the 9th annual ACM symposium on User interface software and technology*, pages 79–80. ACM.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. (2009). ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, volume 3.

Robles-De-La-Torre, G. (2006). The importance of the sense of touch in virtual and real environments. *IEEE Multimedia*, 13(3):24–30.

Rowan, D. (2010). Kinect for Xbox 360: The inside story of Microsoft's secret 'project natal'.

Rusu, R., Foote, T., Mihelich, P., and Wise, M. (2010). ROS Kinect. `http://www.ros.org/wiki/kinect`.

Schröder, Y., Scholz, A., Berger, K., Ruhl, K., Guthe, S., and Magnor, M. (2011). Multiple Kinect studies. *Computer Graphics*.

Starner, T., Weaver, J., and Pentland, A. (1998). Real-time american sign language recognition using desk and wearable computer based video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1371–1375.

Stoakley, R., Conway, M., 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*, pages 265–272. ACM Press/Addison-Wesley Publishing Co.

Stowers, J., Hayes, M., and Bainbridge-Smith, A. (2011). Altitude control of a quadrotor helicopter using depth map from Microsoft Kinect sensor. In *2011 IEEE International Conference on Mechatronics (ICM)*, pages 358–362. IEEE.

Sumar, L. and Bainbridge-Smith, A. (2011). Feasability of fast image processing using multiple Kinect cameras on a portable platform. *Department of Electrical and Computer Engineering, University of Canterbury, New Zealand*.

Wobbrock, J., Wilson, A., and Li, Y. (2007). Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*, pages 159–168. ACM.