Universität zu Köln

Masterarbeit

3D-Interaktion und Visualisierung in der Wissenschaft

SAVKiT: Visualization and Manipulation of Stereographic 3-D Objects

Presented by:
Pedro Fernando
Arizpe Gomez
Matrikelnummer:
5500958

Professor:
Prof. Dr. Lang
Supervisor:
Dipl.-Inf. Daniel
Wickeroth

17.05.2016

Contents

1 Introduction

The main purpose of this project is to implement a user interface for the visualization of 3-D objects on a smart 3-D-TV, through a stereoscopic representation, and to enable their manipulation through the integration of the functionality of a PQLabs G4Series touchscreen, and a Kinect 2.0 camera.

2 State of the Art

This section describes the current state of the science regarding stereoscopic 3-D visualization, manipulation and Natural User Interfaces (NUIs)

In his book "A brave NUI World" [1], Wigdor makes a clear differentiation between the possibilities that specialized hardware can offer and the factual naturality of the interaction. He states that the hardware itself doesn't define or guarantee a NUI, it just gives the opportunity to create one.

Moreover, the characterization as NUI depends on the design of the interface and its capability of adapting and applying new technologies to utilize human capabilities and empiric experience to apply adequately to the given contexts and tasks.

This differentiation is very important when attempting to create a NUI, since the hardware is just a tool that allows for better gesture implementation, but the adequate application and the implementation of intuitive gestures that ensure a natural functionality is indispensable.

2.1 Previous Theses

This subsection describes the work previously done by other students who also implemented and explored multi-touch and free hand gesture interaction for virtual reality (VR).

2.1.1 UTouch3D

Baumesberger designed and implemented a self-designed multi-touch screen that can display 3-D virtual objects through stereoscopy. He called it uTouch3D [2, 3].

The touchscreen of the uTouch3D can theoretically track an unlimited number of fingers simultaneously. This feature offers the user a further possibility of interaction with virtual objects. UTouch3D's user interface exploits this possibility and allows the user to manipulate the visualization.

The original version of the program [2] is designed to allow the user to select, rotate, scale and translate the objects in the visualization through touch gestures. The user can switch between these interactions by changing the amount of fingers in the screen or the duration of contact of a single finger.

Despite its advantages, the uTouch3D also had a downside, since the touchscreen movements are still limited to a two dimensional plane and the displayed objects are in three dimensions, which makes it necessary to implement complex gestures to compensate the missing dimension.

Furthermore, the finger tracking mechanism encountered some difficulties regarding the thickness of every individual finger and continuity of the tracking, thus limiting the factual number of simultaneously detected fingers and restraining the theoretical functional possibilities.

Delisavas [3]enhanced the functionality of uTouch3D by incorporating free hand movements into the interface. These movements are read by a kinect camera 1.0. He modified the interface to accept touch, free hand and combinations of both inputs. He also exploited the hand tracking function of the camera to improve the tracking accuracy of the touch screen.

Nonetheless, the technical limitations of this camera narrowed the interaction possibilities. For example, that version of the camera could not yet quickly recognize if the hand was open or closed without excessive modifications and unreasonable computational effort.

2.2 Omicron project and SAGE2

The Omicron project is a software from the Electronic Visualization Laboratory of the University of Illinois in Chicago. It is currently implemented in the CAVE2 environment, which combines features of the first version of the CAVE; a multiwall, stereoscopic visualization's environment; and tiled display walls (TDW).

Omicron can receive data from several input types, including touch overlays and motion tracking system. It is specially interesting because the touch overlays interface is based on the same touchscreen hardware as the project of this work, namely PQLabs Touchscreen.

This software specializes in the integration of several input devices. Then it streams data from these heterogeneous devices in such a manner that the SAGE2 server can interpret [4]. The software described in this work contains modules that serve this same purpose. Both projects allow the user to decide on his method of input, and hereby they offer more possibilities of interaction.

However, the SAGE2 project focuses rather on the web-based collaboration than the visualization and manipulation of objects.

2.3 Kinoogle

Kinoogle [5] is a natural interaction interface for Google Earth using Microsoft Kinect. It allows the user to control google earth through hand and full body gestures.[1] The user can pan, zoom, rotate, tilt and jump to street view by using predefined body gestures that are conceptually similar to the gestures used by this project's software. (see subsection C.2)

2.4 Other software

The development of new software and hardware for 3-D object visualization and manipulation has been recently boosted by the appearance of head

[1] As this Kinoogle paper was written, only Kinect 1 camera was available in the market, therefore, the hand state function was not fully available

mounted devices (HMDs), like Oculus-Rift as an example for submersive virtual reality or Google Glasses and HoloLens for hybrid reality.

Fluke and Barnes' (2016) [6] paper, from the SAO/NASA Astrophysics Data System, regarding the ultimate display, ponders whether 3-D data visualization in immersive stereoscopic virtual reality in form of the CAVE2 or the Oculus-Rift would help astronomers to better process captured data.

However, they conclude by acknowledging the flaws and shortcomings of both:

> As a low-cost, highly portable solution, HMDs remain an intriguing option. From the perspective of display quality, HMDs clearly have a long way to go before they match either the visual acuity or binocular disparity limit. It seems very unlikely that there will be orders of magnitude changes here in the short term.
>
> A CAVE2 configuration can reach the visual acuity limit and fill the horizontal FOV, but does not fare quite so well with vertical coverage. [6]

3 Motivation and Problem Statement

3.1 Motivation

This section explains the motivation behind this project. As early as 1988, computer scientists recognized the importance of simple direct manipulation controllers. Chen et al.(1988) also outlined some application possibilities:

> "Currently, simple direct manipulation controllers do not exist for 3-D object positioning. The design of such controllers could be important interface contributions for application environments such as manufacturing, architecture, and engineering design, which rely heavily on the display and control of three dimensions." [7]

This statement elucidates the importance of the creation of newfangled input methods or even user interface concepts. These should give the

user the feeling of continuity between his empiric experience of object manipulation and his interaction with the virtual objects. These characteristics equate those of a NUI.

In Wigdor's words, [1] Users naturally develop associations between what they want to do and where they do it from memory-triggered context. This delineates the core of the problem statement.

The essence of this project lies in the challenge of expanding the user's real world experience to the manipulation of virtual objects through natural movements. Notwithstanding that industry standards dictate most design canons for user interfaces, the pursuit of neoteric intuitive interaction possibilities is still worthwhile.

3.2 Problem Statement

Virtual three-dimensional objects originating from design or 3-D scans are becoming increasingly common. As stated by Yuan et al. [8], "The recent proliferation of stereoscopic three dimensional (3D) video technology has fostered a large body of research into 3D video capture, production, compression and delivery. However, little research has been dedicated to the design practices of stereoscopic 3D video interaction". Therefore, a need for adequate visualization manipulation possibilities has emerged.

The specific purpose of this project is the design of a software that can cope with these visualization manipulation necessities taking advantage of several input possibilities.

The concrete target scenario of this project is the visualization of a single virtual object.

The main utilization potentiality of this project is the public display in museums and similar interaction environments, where the public can get a better and more interactive visualization experience.

Another utilization opportunity is found on the didactic. Educational facilities could use this software as a tool for interactive learning, offering pupils an enhanced learning process.

For example, this software can display anatomically correct organs as a stereogram, which would help life sciences students to assimilate better

the visual information and expedite the learning process. As opposed to 2-d schemes and pictures, stereograms provide a sense of depth and the possibility to observe the object from all angles.

For diagnosis purposes, this program could also be a great supporting tool for medical professionals, allowing for an effective and non invasive method of analysis.

3.3 MoSCoW

The functional requirements set for this project can be categorized according to the MoSCoW [9] prioritization method. This method suggests a differentiation between what the program must, should, could and won't do.

There has been some criticism regarding this technique, due to its lack of precision concerning the guidelines for categorization. Wiegers [10] states that this method lacks analysis of importance and urgency. However, for the purposes of this work, the consideration of forthcoming releases or a development time-box is unimportant, and therefore MoSCoW remains a suitable method.

Naturally, most of the requirements for this software have been inherited from Laurentius work [11]

3.3.1 What must the program do?

1. The program **must** be able to display a 3-D virtual object through stereoscopy.
2. The program **must** be able to interpret 3-D scans in form of polygon and stereolitography file types.
3. The program **must** accept three different types of input: mouse/keyboard, touchscreen and free hand movements.
4. The user, using any of the three input methods **must** be able to rotate the displayed object and the camera in all directions, giving 3 DOF for rotation.

5. The user, using any of the three input methods **must** be able to translate the displayed object in all directions, giving 3 DOF for translation.
6. The visualization **must** be in perspective mode.
7. The program **must** display a real-time visual feedback for all manipulation interactions.
8. The program **must** give the option to display the object with or without color information from PLY files.
9. The program **must** provide lighting by four spot lights whose intensity and position can be adjusted in all three spatial axes to generate the desired shadows, which are necessary for optimal visibility of the details of the object.
10. The program **must** allow the user to save the view configuration of the object. This concerns in particular the rotation of the object and the camera; the zoom factor; the orientation and parameters of the lamps.
11. The program **must** allow the user to rotate the camera exactly by one degree with special buttons to enable the capture of the usual 6-fold view (front, rear, left, right, up, down) for scientific publications.
12. The software **must** be open source and the source code must be made accessible and can be changed, which is ensured by its own license, but also by that of the included libraries.
13. The code **must** be sufficiently commented to allow future expansion by any other programmers.

3.3.2 What should the program do?

1. The program **should** be able to display the width, height and thickness of the object in millimeters.
2. The program **should** give the user the option to display overlapping layers of the object with additional information. For example, automatically detected edges or depressions of the object.

3. The program **should** give the option to tack a note at any point on the surface of the displayed object. Its textual content stands on the left edge of the window and a line connects the note to the point on the surface to which the annotation is associated.

4. The program **should** give the option to display or hide any note texts, if any exist.

5. The program **should** be able to store the contents of the 3D display area as an image file at all times (Screenshot).

3.3.3 What could the program do?

1. The program **could** include gestures that isolate interaction types, for example, the user could scale the object with a specific gesture.

2. The program **could** include a status bar to report the current state of interaction to the user.

3.3.4 What won't the program do?

1. The program **won't** include gestures that require a combination of input types.

2. The program **won't** not read nor save the current position of the table.

3. The user **won't** receive tactile feedback.

4. The program **won't** include tracking for wearable pointers.

5. The program **won't** offer the possibility to alter the object's nodes nor edges.

The implementation of gestures that combine several input types has been excluded to maintain the smoothness of the simulation and to avoid tracking inconsistencies between input devices. The position of the table and the referencing of hand and body positions relative to it has been intentionally left out to reduce error proneness and to allow the user to start the interaction anywhere.

4 Technical background and primordial software

4.1 Definitions

Due to the fact that throughout the literature not all definitions are uniform, the definition of some central concepts need to be specified for the specific purposes of this work.

Direct Manipulation: Back in the 70's, direct manipulation was understood as the possibility to interact "directly" with the windows on the screens using the mouse. Actions such as rotating, dragging and moving would be considered as direct manipulation.

In the 80's, Ben Shneiderman from the University of Maryland [12] cited display editors, primitive video games and even programming of industrial robots to replicate human actions as examples for direct manipulation.

Later on, as the first touch screens appeared, this concept evolved in meaning, redefined to be the interaction with objects "directly" underneath the fingers.

Considering this, the definition of direct manipulation used in this work is:

> An interaction between the user and the visualization that translates the movements of the first directly into an appropriate and proportional transformation of the second with immediate visual feedback.

Visualization: Foley and Ribarsky suggest a definition of visualization that coincides with the connotation of this work: [13]

> A useful definition of visualization might be the binding (or mapping) of data to a representation that can be perceived. The types of binding could be visual, auditory, tactile, etc. or a combination of these

Gesture In accordance to George and Blake, the definition of gesture referred to in this work is: "Gestures are metaphors for discrete, indirect, intelligent

interaction" [14]. More specifically, free hand gestures are defined as specific body positions and signals that can be interpreted by the hardware and translated into an information stream that triggers certain procedures within the simulation.

4.2 Geometry and 3-D visualization

The typical human-computer-interaction takes place through 2-D displays, therefore an optical illusion is necessary in order to visualize 3-D virtual objects and give the impression of depth. This illusion is called a stereogram.

4.2.1 OpenGL

Monitors and projectors can only display a finite number of picture elements (pixels) per frame. Therefore, many homogeneous transformation, projection and clipping operations are necessary to enhance the visualization experience. Hereby, a graphics library becomes an essential tool for visualization.

OpenGL enables a simplified implementation that includes all the necessary mechanisms to deal with geometry; through a visualization frustum, stereoscopy i.e. stereopsis, projection, perspective, coloring, opaqueness, shading, and buffering; in the form of predefined concepts, operations and variables inside a graphics library.

4.2.2 Homogeneous transformations

Let $\mathbb{M} \in \mathbb{R}^{4\times4}$ be the Model Transformation Matrix. This Matrix is a homogeneous matrix that contains all executed transformations of the visualized object. namely all $\mathcal{R} \in \mathbb{R}^{4\times4}$ for rotation matrices, $\mathcal{S} \in \mathbb{R}^{4\times4}$ for scaling matrices and $\mathcal{T} \in \mathbb{R}^{4\times4}$ for translation matrices. For the purposes of this project, $\mathbb{M}$ will be called the Artifact Matrix. [15]

Now let $\mathbb{V} \in \mathbb{R}^{4\times4}$ be the View Matrix. This Matrix is also a homogeneous matrix that maps all the world coordinates obtained by the previous transformation to camera coordinates.

Finally, let $\mathbb{P} \in \mathbb{R}^{4\times4}$ be the Projection Matrix. Since the main purpose of this project is to visualize objects and not to realize measurements,

the most suitable choice for $\mathbb{P}$ is a perspective transformation.

The successive multiplication of $\mathbb{M}, \mathbb{V}$ and $\mathbb{P}$ results on the MVP matrix, which can be processed very efficiently by the graphic processing unit (GPU) giving room to a smooth and continuous visualization throughout the interaction.

4.3 Artifact Viewer

This subsection describes the original structure, functionality and form of the Artifact Viewer program (AV) which was the primordial[2] software of this project [11]. The AV is a visualization software that can read polygon (PLY) and STereoLithography (STL) files. The objects were displayed inside a Qt-based GUI on a monochromatic or color gradient background.

The original conception of the Artifact viewer was done with the purpose of devising a tool that could display archaeological objects in 3-D and give users the opportunity to manipulate several aspects of the visualization and also carry out measurements and annotations.

However, this program still maps the object into a two-dimensional representation and the input possibilities were limited to keyboard and mouse.

On the side bar the user has the possibility to interact with the configuration of the visualization and some mouse and keyboard events allowed further manipulation of the object.

4.4 Structure and Functions

4.4.1 File compatibility

AV is able to read from two types of files, *.stl and *.ply. This is managed by two plugins that can interpret the data on the files and transform it into an object.

[2]Primordial: That constitutes the origin or starting point from which something else is derived or developed, or on which something else depends; fundamental, basic; elemental. [16]

4.4.2 Menu

The top screen menu offers pre-programmed tasks in four categories: file, tools, options and help. The **file** menu includes the options to open a new object in the visualization area and to save the current configuration of the visualization into an XML file. The **tools** menu included only the option to make a screenshot of the current state of the visualization. In the **options** menu, the user can change the background color and, in case the vertex colors usage is deactivated, the user can select a color for the artifact. In the **help** menu, the about button offers information about the current version and the author, and the help button opens the documentation of the program.

4.4.3 3-D Visualization panel

The visualization panel is the main part of the widget. Whenever a file is open, the virtual object is displayed.

4.4.4 Lighting

On the side menu, the user has the possibility to configure the lighting of the object. The shading model is based on "Phong shading". First of all, the first checkbox on the top left of this section, "use lightning" toggles the activation of lightning. The second one: "lights visible", on the top right, toggles the visibility of of the lights ant the rotation circumference of the selected light. The program has four light sources that can be rotated around the object along these two circumferences.

If "lights visible" is on, lights are displayed as small lamps in pyramid form. The drop-down menu under the "lights" title is used to select one of the four lights and enable its rotation and intensity and distance modifications.

4.4.5 Rotation

The next item on the sidebar allows the user to make a controlled rotation of the camera by the selected amount of degrees in the selected direction (up, down, left, right) or of the artifact, clockwise or counterclockwise.

4.4.6 Visual Enhancements

The visualization can be modified, independently from lighting, to darken or brighten edges and to darken valleys.

4.4.7 Annotations and Points

The user can select a point in the surface of the object and create an annotation, if there are two selected points, he can measure the distance between these points. If exactly three points are selected, the program can measure the angle between the line connecting the first and second point and the line connecting the second and third points. If three or more points are selected, the program offers the possibility to calculate the convex surface connecting these points.

4.4.8 Mouse and Keyboard event handling

The AV can receive mouse, keyboard and combined input, for example if the shift key is pressed, the user can move the position of the camera by clicking and dragging the mouse. Without the shift key modification, the left mouse click is used for points and annotations and the right one to rotate the object.

4.4.9 Keyboard shortcuts

Some keyboard shortcuts are implemented to facilitate manipulations like toggling lighting and rotating

5 SAVKiT

This section describes the ideological conception of the program and presents the modifications from the original AV program that lead to the Stereoscopic Artifact Viewer with Kinect and Touch (SAVKiT).

5.1 Concept

The main idea behind SAVKiT is the implementation of a NUI that allows the user to execute all homogeneous manipulations described in 4.2.2, namely rotation, scaling and translating through several input methods, allowing the user to choose the most adequate for his intention.

A very important characteristic of NUI is the intuitiveness of the movements. On touch screens, some movements have pushed through as industry standards, like pinch to shrink and split to enlarge. However, 3-D direct manipulation is still developing. Therefore, gestures and movements are also still maturing. Nevertheless, the movements and free hand manipulation possibilities considered in this project seem to be in accordance to other previous works, like Kinoogle [5], as mentioned in subsection 2.3.

The program is designed to give the user the impression of being able to grab the object in the air and manipulate it in different ways, allowing for an intuitive translation of movements into visual feedback.

5.2 Stereoscopy Implementation

The first modification of the original program was the use of Quad-buffering to enable active stereoscopic display of the visualized objects (See A.1). For this purpose, the introduction of parallax, achieved through a second camera, was necessary, which also implied the duplication of all homogeneous transformations and projections.

5.3 New Keyboard Shortcuts

According to Shneiderman's 8 golden rules [17], designers should cater to universal usability, allowing frequent users to expedite routine tasks and have better control over the interface.

Key Combination	Action
Ctrl+O	Open File
Ctrl+S	Save File
Ctrl+F	Toggle Full-screen[3]
Ctrl+P	Screenshot
Ctrl+M	Show Mouse
Ctrl+H	Hide Mouse

5.4 Touch Screen Enhancements

5.4.1 Smart 3-D-TV Touchscreen Gestures

This subsection contains the gestures that AV can interpret from the touchscreen. The kind of manipulation depends on the number of fingers on the screen.

When a first finger touches the screen, the touchscreen coordinates are mapped to the widget coordinates where the object is displayed. If the mapped coordinates are inside of the widget, the touchscreen interaction begins. Successive fingers activate other functions, and if a finger leaves the screen, the interaction adapts to the new quantity of fingers on the touch screen. If no fingers are left, the touch screen interaction comes to an end.

One finger interaction

If there is only one finger on the screen, then the user can change the position of the camera along the X and Y axes, giving the impression that the whole scenario is moving in the same direction as the finger on the screen.

Let $F_{1,t} = \begin{pmatrix} x_{1,t} \\ y_{1,t} \end{pmatrix} \in \mathbb{R}^2, t \in \mathbb{Z}$ be the initial finger position at frame t with (x, y) coordinates $(x_{1,t}, y_{1,t})$ and let

$$\Delta F_{1,t} = \begin{pmatrix} \Delta x_{1,t} \\ \Delta y_{1,t} \end{pmatrix} = \begin{pmatrix} x_{1,t} - x_{1,t-1} \\ y_{1,t} - y_{1,t-1} \end{pmatrix}$$

be the finger displacement. Now let $\mathcal{O}c_t$ be the origin of the camera at frame

t. Then, the movement of the camera is calculated by:

$$\mathcal{O}c_{t+1} = \mathcal{O}c_t + \begin{pmatrix} \frac{-\Delta x}{6.3} \times \frac{CDO}{150} \\ \frac{\Delta y}{6.3} \times \frac{CDTO}{150} \end{pmatrix}$$

where CDO is the distance from the current camera position to the origin.

Two-finger interactions

If there are two fingers on the screen and the distance between them is increased, then the camera goes closer to the object, if the distance is reduced, then the camera goes further from the object, giving the impression of zooming in and out, respectively. If the line between the fingers is rotated, the object will rotate around the axis that is orthogonal to the screen plane.

Let $F_{i,t} = \begin{pmatrix} x_{i,t} \\ y_{i,t} \end{pmatrix} \in \mathbb{R}^2, t \in \mathbb{Z}$ be the position of finger i at frame t with (x, y) coordinates $(x_{i,t}, y_{i,t})$. The distance between fingers at frame t $d_{t1,2}$ is calculated as follows:

$$d_{t_{1,2}} = \|F_{1,t} - F_{2,t}\|_1$$

where $\|\cdot\|_1$ is the l_1 norm, since its calculation is significantly more efficient than the euclidean distance. Then, the zooming operations are executed by the following formula:

$$CDO_t = CDO_{t-1} \cdot \frac{d_{t-1_{1,2}}}{d_{t_{1,2}}}$$

Now let $\overrightarrow{v_t} = F_{2,t} - F_{1,t}$ be the line between fingers at frame t. Then, the angle of rotation $\theta_{touch} \in [0, 360]$ around the z axis is calculated by:

$$\theta_{touch} \frac{-180}{\pi} \cdot \arccos(< \overrightarrow{u}, \overrightarrow{v} >)$$

Three-and-more-finger interaction

If there are three or more fingers on the screen and they are moving in the same direction, a simulation of the trackball movement is initialized.

Depending on the mode set for the trackball, the object follows a 3-D rotation as a sphere or in the plane. The speed of the trackball movement is divided by the number of fingers on the screen.

Let $F_{i,t}$ be defined as above. Let n be the number of fingers touching the screen, then the position of the pointer of the trackball movement is determined by:

$$\sum_{i=1}^{n} \frac{1}{n} F_{i,t}$$

5.5 Kinect 2 Camera Gesture implementation

Most current video games and applications for Kinect generate a pointer as an interaction dummy for the virtual world. SAVKiT uses relative position data, which means that the user can start interaction anywhere in the kinect camera space and the movement will be calculated from the relative position of the hands from frame to frame.

The program receives the coordinates of the hand position given by the kinect camera and transforms them to fit the visualization coordinates' orientation. Since the screen is used in an horizontal position, the following transformation is required:

$$\mathcal{F} : \mathbb{R}^3 \to R^3;$$
$$\mathcal{F} : \begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} x \\ -z \\ y \end{pmatrix}$$

where $\mathcal{F}$ is applied to all coordinates received from the kinect camera in order for them to match the coordinates on the visualization environment.

The Kinect Camera 2.0 has the capability of recognizing the state of the hands. It can differentiate between a closed, open and lasso positions. (see Figure 2).
In their Microsoft Kinect natural user interface for Google Earth navigation, Boulos et al. [5] distribute the three visualization manipulation possibilities; namely rotate, scale and translate (RST); among two free hand gestures. Rotate and scale are executed with two closed hands and translation (pan) is done with one hand open and the other one closed. The interaction design of this project follows this example and extends it to isolate the scaling

Figure 1: Kinect Hand States depiction

interaction into a third hand state, i.e. lasso, and offers two possibilities for translation.

In addition, the attempts to implement all three visualization manipulations into one gesture resulted in an unstable interaction with faulty visual feedback and great error susceptibility, despite the utilization of a Kalman filter.

Kinect interaction can be activated by holding both hands open for two seconds. Once the interaction is initialized, the following manipulations are possible:

5.5.1 Rotate

Both hands closed for 2 seconds start the rotation, until one or both hands are opened. The rotation is programmed with 6 degrees of freedom (DOF).

The angle of the rotation is calculated by creating a vector between the point coordinates of both hands and then comparing its angle with the vector of the next frame by means of a dot product.

Let $R_t \in \mathbb{R}^3$ represent the coordinates of the right hand at frame $t, t \in \mathbb{Z}$ and $L_t \in \mathbb{R}^3$ represent the coordinates of the left hand at frame t.

Then, the **angle of rotation** $\theta_{kinect} \in [0, 360]$ in degrees is given by:

$$\theta_{kinect} = \frac{180}{\pi} \cdot \arccos(< \overrightarrow{u}, \overrightarrow{v} >)$$

And the **axis of rotation** by

$$Axis = \overrightarrow{u} \times \overrightarrow{v}$$

where

$$\overrightarrow{u} = \frac{R_{t-1} - L_{t-1}}{\|R_{t-1} - L_{t-1}\|} \ , \ \overrightarrow{v} = \frac{R_t - L_t}{\|R_t - L_t\|}$$

. At frame $t = 0$ the initial positions of the hands are captured, therefore these formulas are valid for $t \geq 1$.

To ensure a smooth interaction, all rotations with an angle of more than 10 degrees are ignored. Furthermore, the axis of rotation remains befitting to the visualization axes, meaning that previous rotations of the artifact matrix do not interfere with the direction of rotation. (see Appendix A.2, line 28)

5.5.2 Scale

If both hands have one finger up, recognized as lasso position by the kinect, for more than 2 seconds, the program initializes the scaling mode. The artifact can be proportionally scaled by changing the separation between hands while keeping the hand position.

Let R_t and L_t be defined as in C.2.1. Then, let D_t be the separation between the hands at frame t, defined by $D_t = R_t - L_t$. Herewith, the scaling factor $s_t \in \mathbb{R}$ between frames is calculated by:

$$s_t = \frac{D_{t-1}}{D_t}, t \geq 1$$

Once again, to ensure the smoothness of the rotation and to avoid abrupt changes, scaling factors outside of the tolerance borders are ignored, meaning that only if $0.75 < s_t < 1.5$, the scaling takes place.

5.5.3 Translate

The left hand up and open, the right one closed for 2 seconds starts a homogeneous translation of the model matrix, whereas the right hand's change of position determines the translation of the artifact, and the left one works as the "signal", meaning that the movement stays on as long as the left hand remains open, or until the right hand opens

Let R_t and L_t be once again defined as above. The translation vector $\Delta_r \in \mathbb{R}^3$ applied to the Artifact Matrix is determined by:

$$\Delta_r = R_{t-1} - R_t, t \geq 1$$

Anew, all previous rotations of the Artifact Matrix are ignored to guarantee an intuitive interaction. The object moves always in the same direction as the hand of the user.

The right hand up and open, the left one closed for 2 seconds starts a homogeneous translation of the projection matrix, where the left hand's change of position determines the translation of the projection, and the left one works as the "signal", meaning that the movement stays on as long as the right hand remains open, or until the left hand opens.

The translation vector $\Delta_l \in \mathbb{R}^3$ applied to the projection Matrix is determined by:

$$\Delta_l = L_{t-1} - L_t, t \geq 1$$

Similar to the model translation, the projection follows the movement direction of the hand.

5.5.4 Untracked or unknown states

If the position of both hands has not been tracked for 5 seconds or if the state of the hand has been unknown for 10 seconds, then the interaction with the kinect camera stops.[4]

[4]unknown and "not Tracked" states just report approximate coordinates

6 Evaluation

The evaluation of this project was in form of a consultation and survey of 10 experts to assess the functionality, usability and intuitiveness of the program. A template of this survey can be found on Appendix B [18]

7 Results

This section presents the results of the previously mentioned survey.

8 Concluding Remarks and future work

This section includes a small summary of the thesis, states adequate conclusions and suggests possibilities for further investigation of the topic.

References

[1] G. Goth, "Brave nui world," Communications of the ACM, vol. 54, no. 12, pp. 14–16, 2011.

[2] A. Baumesberger, "utouch3d: Multi-touch interaction in virtual reality," Universität Koblenz Landau, Diplomarbeit, 2010.

[3] S. Delisavas, "Freihändige gesteninteraktion und multi-touch in virtual reality umgebungen," Universität zu Köln, Diplomarbeit, 2013.

[4] T. Marrinan, J. Aurisano, A. Nishimoto, K. Bharadwaj, V. Mateevitsi, L. Renambot, L. Long, A. Johnson, and J. Leigh, "Sage2: A new approach for data intensive collaboration using scalable resolution shared displays," in Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2014 International Conference on, pp. 177–186, Oct 2014.

[5] M. N. K. Boulos, B. J. Blanchard, C. Walker, J. Montero, A. Tripathy, and R. Gutierrez-Osuna, "Web gis in practice x: a microsoft kinect

natural user interface for google earth navigation," International journal of health geographics, vol. 10, no. 1, p. 1, 2011.

[6] C. J. Fluke and D. G. Barnes, "The Ultimate Display," ArXiv e-prints, Jan. 2016.

[7] M. Chen, S. J. Mountford, and A. Sellen, "A study in interactive 3-d rotation using 2-d control devices," in Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '88, (New York, NY, USA), pp. 121–129, ACM, 1988.

[8] H. Yuan, J. Ćalić, A. Fernando, and A. Kondoz, "Investigation and evaluation of pointing modalities for interactive stereoscopic 3d tv," in Multimedia and Expo (ICME), 2013 IEEE International Conference on, pp. 1–6, July 2013.

[9] S. Ash, "Moscow prioritisation," DSDM Consortium, 2007.

[10] K. Wiegers and J. Beatty, Software requirements. Pearson Education, 2013.

[11] D. M. Laurentius, "Visualisierung archäologischer datensätze," Universität zu Köln, Diplomarbeit, 2014.

[12] B. Shneiderman, "Direct manipulation: A step beyond programming languages," in ACM SIGSOC Bulletin, vol. 13, p. 143, ACM, 1981.

[13] W. Ribarsky and J. D. Foley, "Next-generation data visualization tools," p. 104, 1994.

[14] R. George and J. Blake, "Objects, containers, gestures, and manipulations: Universal foundational metaphors of natural user interfaces," CHI 2010, pp. 10–15, 2010.

[15] Reisman, J. L., Davidson, P. L., Han, and J. Y., "A screen-space formulation for 2d and 3d direct manipulation," in Proceedings of the 22Nd Annual ACM Symposium on User Interface Software and Technology, UIST '09, (New York, NY, USA), pp. 69–78, ACM, 2009.

[16] O. E. Dictionary, "Oed online," Oxford University Press. http://www.oed. com, Accessed Nov, vol. 30, p. 2006, 1989.

[17] B. SHNEIDERMAN, "Designing the user interface: strategies for effective human-computer interaction. boston: Addisonwesley, 2010, xviii, 606 s," tech. rep., ISBN 978-0-321-53735-5.

[18] J. Brooke et al., "Sus-a quick and dirty usability scale," Usability evaluation in industry, vol. 189, no. 194, pp. 4–7, 1996.

Appendix A code

A.1 Stereo format Code

```
QGLFormat stereoFormat;
stereoFormat.setSampleBuffers(true);
stereoFormat.setStereo(true);
this->setFormat(stereoFormat);
```

A.2 Kinect Rotation Code

```
//refresh hand positions
 if(!mano.isLeft)
 {
 m_kNewRightRotPos=QVector3D((qreal)mano.x,
 (qreal)-mano.z,(qreal)mano.y);
 }
 if(mano.isLeft)
 {
 m_kNewLeftRotPos=QVector3D((qreal)mano.x
 ,(qreal)-mano.z,(qreal)mano.y);
 }
 //draw new rotation line
 m_kNewRotVec=m_kNewRightRotPos-m_kNewLeftRotPos;
 //get angle
 m_KRotAngle=acos(QVector3D::dotProduct(
 m_kOldRotVec.normalized(),m_kNewRotVec.normalized()
 ) )* 180.0f / PI
 //get axis
 m_axis = QVector3D::crossProduct(m_kOldRotVec.normalized(),
 m_kNewRotVec.normalized());
// get ScaleFactor
m_kScFactor=m_kNewRotVec.length()/m_kOldRotVec.length();
 //rotate by axis and angle
 if(m_KRotAngle< 30 | | m_KRotAngle>330)
 {
m_MatrixArtefact.translate(m_model->m_centerPoint);
```

```
27 m_MatrixArtefact.translate(m_camOrigin);
28 m_axis = m_MatrixArtefact.inverted().mapVector(m_axis)
29 m_MatrixArtefact.rotate(m_KRotAngle,m_axis.normalized());
30 m_MatrixArtefact.scale(m_kScFactor);
31 m_MatrixArtefact.translate(-m_camOrigin);
32 m_MatrixArtefact.translate(-m_model->m_centerPoint);
33  }
34
35  updateGL();
36  //update hand positions and connection vector
37  m_kOldLeftRotPos=m_kNewLeftRotPos;
38  m_kOldRightRotPos=m_kNewRightRotPos;
39  m_kOldRotVec=m_kNewRotVec;
```

Appendix B Expert survey template

Gender:M/F Age:____ Occupation (Major for students):_____

1. I think that I would like to use this system frequently

1 (totally disagree)	2	3	4	5 (totally agree)

2. I found the system unnecessarily complex

1 (totally disagree)	2	3	4	5 (totally agree)

3. I thought the system was easy to use

1 (totally disagree)	2	3	4	5 (totally agree)

4. I think that I would need the support of a technical person to be able to use this system

1 (totally disagree)	2	3	4	5 (totally agree)

5. I found the various functions in this system were well integrated

1 (totally disagree)	2	3	4	5 (totally agree)

6. I thought there was too much inconsistency in this system

1 (totally disagree)	2	3	4	5 (totally agree)

7. I would imagine that most people would learn to use this system very quickly

1 (totally disagree)	2	3	4	5 (totally agree)

8. I found the system very cumbersome to use

1 (totally disagree)	2	3	4	5 (totally agree)

9. I felt very confident using the system

1 (totally disagree)	2	3	4	5 (totally agree)

10. I needed to learn a lot of things before I could get going with this system

1 (totally disagree)	2	3	4	5 (totally agree)

Thank you very much for your input.

Appendix C Survey Handout

SAVKiT: Stereographic Artifact Viewer with Kinect and Touch

C.1 Touchscreen

One finger on the screen moves the camera away from your finger, giving the impression that you are moving the world.
Two fingers on the screen activate zooming and z-axis rotation simultaneously. Pinch to zoom and turn to rotate.

If there are three or more fingers on the screen and they are moving in the same direction, a simulation of the trackball movement is initialized. Depending on the mode set for the trackball, the object follows a 3-D rotation as a sphere or in the plane. The speed of the trackball movement is divided by the number of fingers on the screen.

Let $F_{i,t}$ be defined as above. Let n be the number of fingers touching the screen, then the position of the pointer of the trackball movement is determined by:

$$\sum_{i=1}^{n} \frac{1}{n} F_{i,t}$$

C.2 Kinect 2 Camera Gesture implementation

Most current video games and applications for Kinect generate a pointer as an interaction dummy for the virtual world. SAVKiT uses relative position data, which means that the user can start interaction anywhere in the kinect camera space and the movement will be calculated from the relative position of the hands from frame to frame.

The program receives the coordinates of the hand position given by the kinect camera and transforms them to fit the visualization coordinates' orientation. Since the screen is used in an horizontal position, the following

transformation is required:

$$\mathcal{F} : \mathbb{R}^3 \to R^3;$$
$$\mathcal{F} : \begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} x \\ -z \\ y \end{pmatrix}$$

where $\mathcal{F}$ is applied to all coordinates received from the kinect camera in order for them to match the coordinates on the visualization environment.

The Kinect Camera 2.0 has the capability of recognizing the state of the hands. It can differentiate between a closed, open and lasso positions. (see Figure 2).

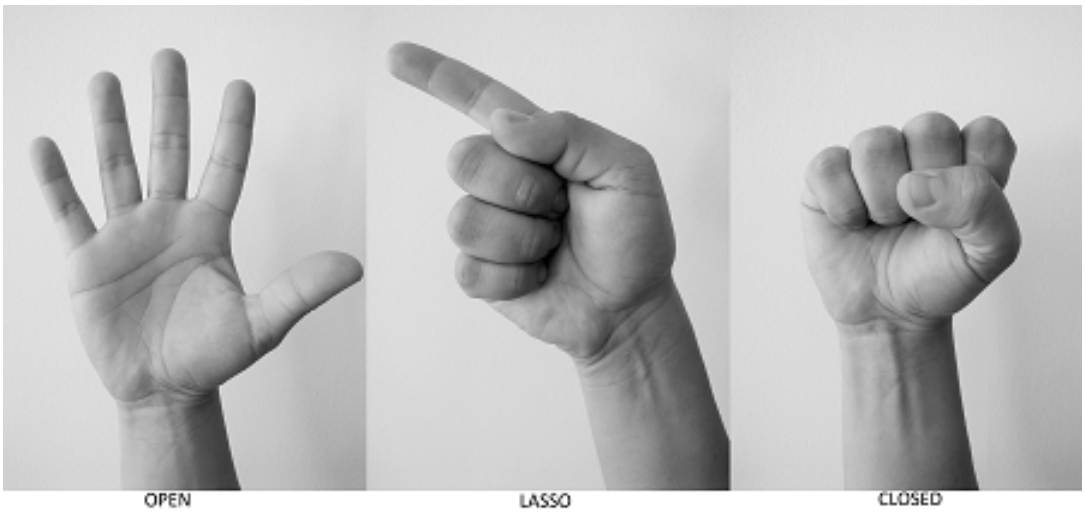

Figure 2: Kinect Hand States depiction

In their Microsoft Kinect natural user interface for Google Earth navigation, Boulos et al. [5] distribute the three visualization manipulation possibilities; namely rotate, scale and translate (RST); among two free hand gestures. Rotate and scale are executed with two closed hands and translation (pan) is done with one hand open and the other one closed. The interaction design of this project follows this example and extends it to isolate the scaling interaction into a third hand state, i.e. lasso, and offers two possibilities for translation.

In addition, the attempts to implement all three visualization manipulations into one gesture resulted in an unstable interaction with faulty

visual feedback and great error susceptibility, despite the utilization of a Kalman filter.

Kinect interaction can be activated by holding both hands open for two seconds. Once the interaction is initialized, the following manipulations are possible:

C.2.1 Rotate

Both hands closed for 2 seconds start the rotation, until one or both hands are opened. The rotation is programmed with 6 degrees of freedom (DOF).

The angle of the rotation is calculated by creating a vector between the point coordinates of both hands and then comparing its angle with the vector of the next frame by means of a dot product.

Let $R_t \in \mathbb{R}^3$ represent the coordinates of the right hand at frame $t, t \in \mathbb{Z}$ and $L_t \in \mathbb{R}^3$ represent the coordinates of the left hand at frame t. Then, the **angle of rotation** $\theta_{kinect} \in [0, 360]$ in degrees is given by:

$$\theta_{kinect} = \frac{180}{\pi} \cdot \arccos(< \overrightarrow{u}, \overrightarrow{v} >)$$

And the **axis of rotation** by

$$Axis = \overrightarrow{u} \times \overrightarrow{v}$$

where

$$\overrightarrow{u} = \frac{R_{t-1} - L_{t-1}}{\|R_{t-1} - L_{t-1}\|} \ , \ \overrightarrow{v} = \frac{R_t - L_t}{\|R_t - L_t\|}$$

. At frame $t = 0$ the initial positions of the hands are captured, therefore these formulas are valid for $t \geq 1$.

To ensure a smooth interaction, all rotations with an angle of more than 10 degrees are ignored. Furthermore, the axis of rotation remains befitting to the visualization axes, meaning that previous rotations of the artifact matrix do not interfere with the direction of rotation. (see Appendix A.2, line 28)

C.2.2 Scale

If both hands have one finger up, recognized as lasso position by the kinect, for more than 2 seconds, the program initializes the scaling mode. The artifact can be proportionally scaled by changing the separation between hands while keeping the hand position.

Let R_t and L_t be defined as in C.2.1. Then, let D_t be the separation between the hands at frame t, defined by $D_t = R_t - L_t$. Herewith, the scaling factor $s_t \in \mathbb{R}$ between frames is calculated by:

$$s_t = \frac{D_{t-1}}{D_t}, t \geq 1$$

Once again, to ensure the smoothness of the rotation and to avoid abrupt changes, scaling factors outside of the tolerance borders are ignored, meaning that only if $0.75 < s_t < 1.5$, the scaling takes place.

C.2.3 Translate

The left hand up and open, the right one closed for 2 seconds starts a homogeneous translation of the model matrix, whereas the right hand's change of position determines the translation of the artifact, and the left one works as the "signal", meaning that the movement stays on as long as the left hand remains open, or until the right hand opens

Let R_t and L_t be once again defined as above. The translation vector $\Delta_r \in \mathbb{R}^3$ applied to the Artifact Matrix is determined by:

$$\Delta_r = R_{t-1} - R_t, t \geq 1$$

Anew, all previous rotations of the Artifact Matrix are ignored to guarantee an intuitive interaction. The object moves always in the same direction as the hand of the user.

The right hand up and open, the left one closed for 2 seconds starts a homogeneous translation of the projection matrix, where the left hand's change of position determines the translation of the projection, and the left one works as the "signal", meaning that the movement stays on as long as the right hand remains open, or until the left hand opens.

The translation vector $\Delta_l \in \mathbb{R}^3$ applied to the projection Matrix is determined by:

$$\Delta_l = L_{t-1} - L_t, t \geq 1$$

Similar to the model translation, the projection follows the movement direction of the hand.

C.2.4 Untracked or unknown states

If the position of both hands has not been tracked for 5 seconds or if the state of the hand has been unknown for 10 seconds, then the interaction with the kinect camera stops.[5]

[5]unknown and "not Tracked" states just report approximate coordinates