3D-Board: A Remote Collaborative Workspace Featuring Virtual 3D Embodiments

Jakob Zillner

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im September 2014

© Copyright 2014 Jakob Zillner

This work is published under the conditions of the *Creative Commons License Attribution–NonCommercial–NoDerivatives 4.0 International* (CC BY-NC-ND 4.0)—see http://creativecommons.org/licenses/by-nc-nd/4.0/.

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere.

Hagenberg, September 19, 2014

Jakob Zillner

Contents

Acknowledgments

The first ideas for *3D-Board* sparked almost two years ago. From that time on Dr. Michael Haller constantly provided support and guidance far beyond the duties of a supervisor. During countless discussions he and the patient colleagues of the Media Interaction Lab always offered an open ear and a helping hand. Gratitude is also due to Dr. Wilhelm Burger for his advice and my fellow students for their help.

With the completion of this thesis, the five-year long chapter of studying comes to an end. There are no words to describe how grateful I am to have a girlfriend that supported me with all her love and wisdom ever since. Never forgotten will be the encouragement of my parents and the inspiration received from my brother. Without all these people this thesis would have never been possible.

Kurzfassung

3D-Board ist ein interaktives Whiteboard-System, dass eine intuitive Zusammenarbeit räumlich getrennter Personen ermöglicht. Werden großflächige Bildschirme für verteiltes Arbeiten eingesetzt, ist die Wahrnehmung der Gestik, Mimik und Interaktion des Gegenübers von besonderer Bedeutung. Daher liegt der Fokus dieser Arbeit auf der effizienten und natürlichen Zusammenarbeit zwischen den verteilten Benutzern. Dies wird erreicht, indem ein lebensechtes, dreidimensionales Abbild der entfernten Person mit der gemeinsam genutzten Arbeitsfläche überblendet wird. Dadurch entsteht der Eindruck, als könnte der Betrachter durch ein transparentes Whiteboard hindurch, in den gegenüberliegenden Raum hinein sehen. Diese neuartige Art der Visualisierung ermöglicht eine intuitive Interaktion mit den Kollaborateuren. Das System wurde in einer Studie evaluiert, welche belegt, dass *3D-Board* die Effektivität von verteiltem Arbeiten auf interaktiven Whiteboards steigert.

Abstract

3D-Board is a telepresence system for interactive whiteboards. When using a large-scale screen for remote collaboration, awareness for the distributed user's gestures and actions is paramount. Thus, the presented system captures life-sized virtual embodiments of a geographically distributed user. By blending the front-facing 3D embodiment of a remote collaborator with the shared workspace, an illusion is created as if the observer was looking through the transparent whiteboard into the remote user's room. This novel visualization technique conveys the full-body pose and gestures to facilitate intuitive cooperation. *3D-Board* was evaluated in a usability study, showing that it significantly improves the effectiveness of remote collaboration on a large interactive surface.

Chapter 1

Introduction

In times of globalization, working remotely becomes evermore important. Computer supported cooperative work is often the only feasible option to exchange with geographically distributed colleagues. An important aspect of remote collaboration is to give the impression that a distant user is present [24, 50, 56]. However, most of the telepresence systems [53] available to date fail to provide the same experience a co-located working environment would offer. One reason is the visual disjunction between the distributed collaborator's representation and the visualization of the shared data [55]. Thus, relating important non-verbal cues of a remote user with the corresponding distributed information often becomes impossible due to the lack of social presence [29]. However, workspace awareness, gestures and facial expressions, are highly relevant for effective collaboration [15, 23, 38].

1.1 3D-Board

This thesis presents *3D-Board*, a remote collaboration system for large interactive whiteboards. The system facilitates social presence in a shared working environment by blending a virtual, front-facing 3D embodiment of the remote user with the digital content of the interactive whiteboard. This novel visualization technique creates the impression as if the remote collaborator would stand behind the shared whiteboard, as depicted in Figure 1.1. Thus, the presented work strives to provide an experience as if all distributed users would work face-to-face on the same interactive, transparent surface. Summarizing, the main contributions of this thesis towards an effective remote collaboration are:

- Enabling remote, face-to-face collaboration on a large, interactive surface. The simple and novel visualization technique facilitates full body, 3D virtual embodiments of the remote user.
- A study that demonstrates the benefits of the remote embodiments. The evaluation shows that the 3D front facing approach improves the

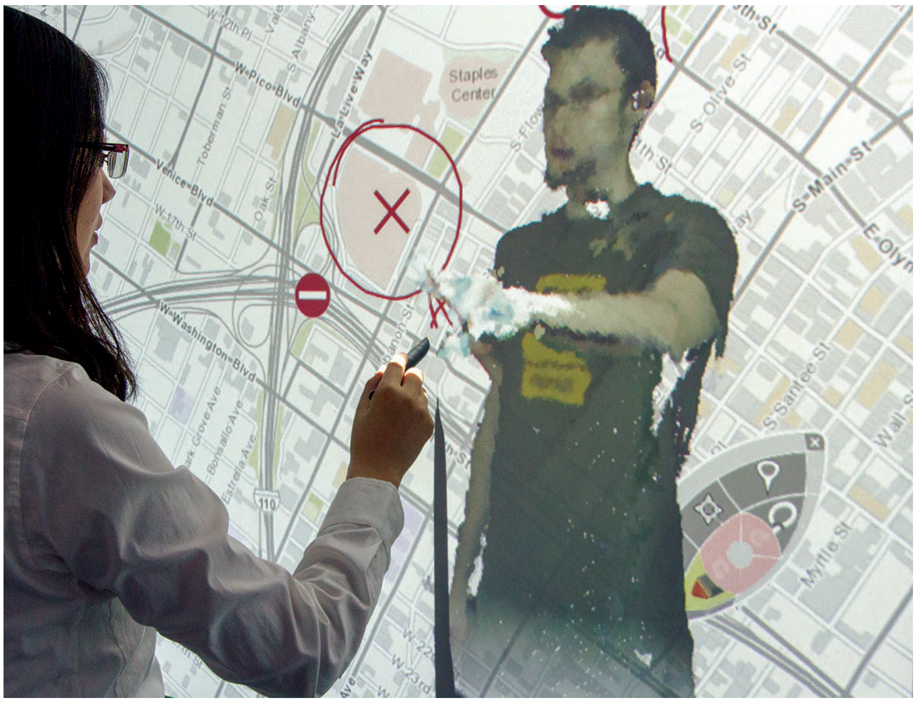


Figure 1.1: *3D-Board* blends the virtual 3D embodiment of the remote user with the digital content of the interactive whiteboard.

user awareness and provides additional features that makes remote collaboration tasks more efficient.

- Robustly segmenting the remote user from the background to blend the visualization with any virtual content and avoid occlusion of important content.
- And finally, featuring a self-contained hardware setup. Since the required cameras can easily be mounted on the edges of the whiteboard and do not have to be behind or in front of the screen, no holographic screens or half-silvered mirrors are needed.

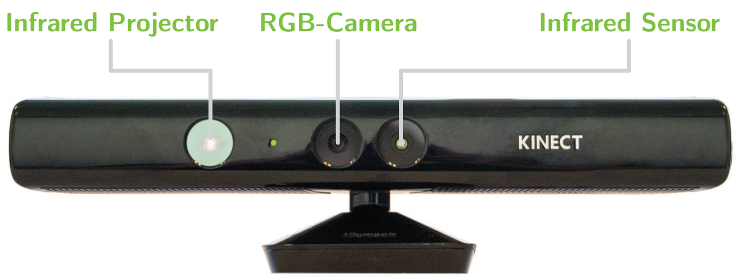
The implementation of *3D-Board*, depicted in Chapter 3 represents an alternative approach to the first version of *3D-Board* [65]. The evaluation, described in Chapter 5, was conducted with the earlier implementation. Both prototypes differ only in their approach to visualize the remote user but feature comparable visual quality. All images presented in this thesis, with the exception of Chapter 3, show the first implementation of *3D-Board*.

1.2 Sensing a remote user

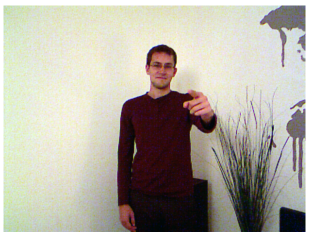
Creating a virtual embodiment of the remote user in 3D requires special depth sensors. *3D-Board* uses multiple Microsoft Kinect v1 sensors (cf. Figure 1.2 (a)) to reconstruct the scene in front of the whiteboard.

The Kinect is a structured light sensor. The infrared emitter projects a structured dot pattern onto an object and the infrared camera captures the projected dots (cf. Figure 1.2 (b)). Based on the displacement of the known pattern, depth positions can be computed for each pixel in the image (cf. Figure 1.2 (c)) [4, 63]. An additional RGB camera captures a color image of the scene (cf. Figure 1.2 (d)) that is used to reconstruct a colored 3D embodiment of the user. In addition, the pose of a person can be estimated from the depth map [52]. The inferred joint positions of that person are utilized by *3D-Board* for tracking the head of an observer. This allows for motion parallax effects to perceive the virtual embodiment in 3D.

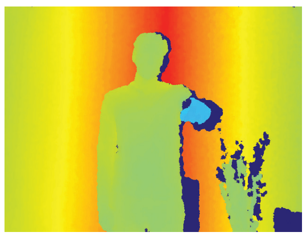
However, when the dot patterns of multiple structured light depth sensors overlap, the sensors cannot determine all of their corresponding dots. This leads to missing depth readings and partly incomplete depth maps. Although hardware solutions exist [10], a software implementation was chosen for handling the interference, as will be shown in Chapter 3.



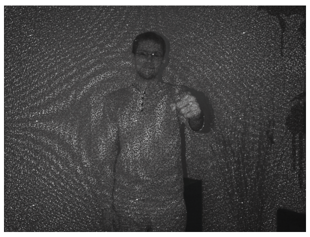
(a)



(b)



(c)



(d)

Figure 1.2: The Kinect v1 sensor (a) outputs the color (b) and the depth image (c) computed from a structured infrared light pattern (d).

1.3 Outline

This thesis covers in detail every aspect of *3D-Board*. At first, the most influential related work is discussed. The conclusions drawn for designing an efficient remote collaboration system are elaborated thereafter. Chapter 3 deals exhaustively with the implementation of the system, utilizing the mentioned depth sensing cameras. After talking about the system's performance and potential applications, Chapter 5 compares *3D-Board* with two other techniques of collaboration. The concluding evaluation highlights the efficiency of the front-facing embodiments blended with the digital content before reasoning about further improvements and final thoughts.

Chapter 2

Inspiration and Concepts

Remote collaboration is a very active field of research. The quantity of literature that influenced the development of *3D-Board* is vast. Thus, the most important findings from related literature will be discussed first, before deducing the concepts.

2.1 Related Work

The main lesson learned from related work is that collaboration efficiency can be raised by raising the social presence of the remote participant.

2.1.1 Social Presence

Social presence defines the degree of salience that a collaborator has in collaborative tasks [5]. The higher the social presence, the more aware a user becomes of the remote partner. Awareness for the common workspace is very important for coordinating actions of all participants. It is defined by Gutwin et al. [24, p. 197] as

> *the up-to-the-moment understanding of another person's interaction with the shared workspace.*

Awareness is influenced by multiple factors such as: the presence and location of another person, the identity of that person, the gaze of the partner as well as the actions and interaction radius of the collaborator [24]. Hence, a remote collaboration system should enable the user to see the participant's face to raise the awareness for facial expressions and gaze [15]. In addition, deictic gestures and gestures in general are highly essential for a distributed conversation [18, 20, 34].

Summarized, a lack of social presence can cause misunderstandings and ineffective teamwork. Therefore, distributed tabletop and whiteboard applications often provide user embodiments to improve the awareness among collaborators [2].

2.1.2 Embodiments

In computer supported cooperative work, the video image is serving as the connection between the distributed users [3]. However, to raise the social presence, the video image and the shared content has to fuse into a single, common workspace [32, 35, 57]. By utilizing virtual embodiments of a remote collaborator the visual disjunction between the collaborator's actions and the actual data is minimized.

Tabletops

The visualization and presentation of the virtual embodiment has a vast influence on the workspace awareness. Different possibilities have mostly been evaluated in the context of tabletops [13, 19, 50, 59].

Doucette et al. [13], for instance, used different levels of abstraction for arm embodiments while interacting with the tabletop. The remote collaborator therefore sees the local user's arms as either just a thin line, a colorized arm, or as a real picture. In the context of a tabletop setup, the results show that the arm visualization, does not significantly raise the awareness for the remote user. The participants did not associate the embodiment with their partner, since they did only see the arms but the rest of the body was missing.

Another evaluation has shown that being able to look the collaborator in the eye raises the social presence [29]. At the same time, the efficiency is lowered due to a higher cognitive load. However, the mental demands might as well be influenced by additional factors like the setup and interaction techniques.

Whiteboards and Vertical Displays

When collaboration over larger scale screens, the presence of a virtual embodiment is highly relevant since the full body of the collaborator is visible. Gaze awareness, facial expressions, and hand pointing play a much bigger role while interacting with another person on such devices.

ClearBoard [31] is one of the most influential setups supporting face-to-face conversations and shared drawing activities on one screen. By utilizing a polarizing film between a projection screen and a half mirror the user can be captured by a camera while drawing on a whiteboard. At the same time the remote collaborator can be seen, thus creating the impression of [31, p. 525]:

talking through and drawing on a transparent glass window

This metaphor was also adopted for *3D-Board* and served as an inspiration.

Based on the ideas of ClearBoard, Gumienny et al. developed Tele-Board [22]. Tele-Board features video- and data-conferencing by overlaying

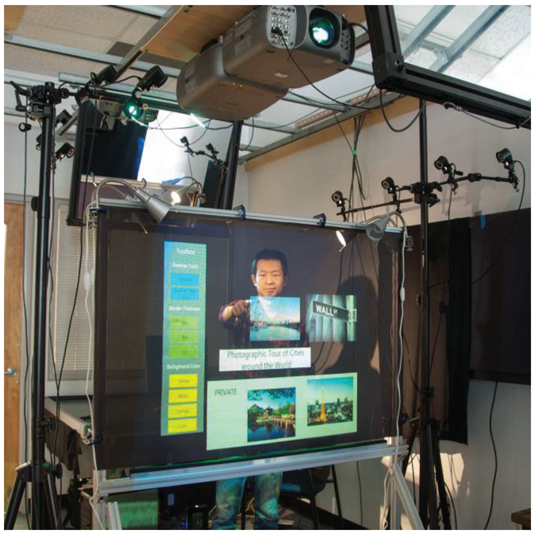


Figure 2.1: The two-sided transparent display allows for face-to-face collaboration. Image source [43].

the transparent content of an interactive whiteboard over the full-screen rear or side video view of the remote collaborators.

CollaBoard [41] takes the idea one step further by separating the user from the background. Thus, the full-sized back-view of the upper body of a user can be superimposed on top of the digital data. While both approaches provide accurate deictic gestures, natural interaction is limited since the face-to-face communication is restricted.

ConnectBoard on the other hand, focuses on gaze awareness [54]. By utilizing a large see-through display and face detection the system supports eye contact by offsetting the video stream based on the viewer's position.

EyeGaze [39] facilitates gaze awareness by creating a 3D model of a person from multiple Kinects and rendering the output from the observer's point of view.

TeleHuman [37] is a videoconferencing system that uses a cylindrical 3D display portal to display a 360° view of a geographically distributed user. By supporting stereoscopic 3D and motion parallax, the system is capable of accurately conveying hand pointing gestures and body poses.

Li et al. [43] concentrate their work on the collaborative interaction with a transparent display. The screen accepts input and can visualize different content on both sides. The shared workspace is projected onto a special fabric. The hole size of the fabric is large enough to see the other collaborator manipulating the shared elements through the projection (cf. Figure 2.1). This face-to-face collaboration concept is very similar to the remote conferencing vision of Corning Inc. [66].

Onespace [42] utilizes a depth-sensing camera to segment users from the background and to get their position in 3D space. Therefore, a 2D representation can be placed in a virtual environment and the distributed partners can collaboratively interact with their digital surroundings. The front-view makes gaze-awareness and gestures possible. However, the depth-sensing camera, mounted on top of the display, forces users to keep a certain distance to the screen. In addition, a mirrored representation of the co-located person needs to be displayed in order to raise the awareness of one's position in 3D space.

Maimone et al. [46] also utilize multiple depth-sensing cameras to capture a scene in 3D. The reconstructed environment can then be viewed by utilizing a head-tracking system and an auto stereoscopic display. This creates the illusion as if the remote user could be seen through a glass window, as envisioned by Ishii's ClearBoard. However, the system does not support collaboration.

Edelmann et al. focus on the interaction with digital content [14]. The system consists of a stereo camera setup behind a semitransparent, holographic, multi-touch screen to capture the remote scene. This allows for a natural, face-to-face collaboration on the interactive surface. However, the need for shutter glasses and the opaque holographic screen disrupts the telepresence experience. In addition, the remote user is not separated from the background, thus the video occludes the shared workspace.

2.1.3 Teleimmersion

Teleimmersion focuses on a credible remote collaboration experience. Most often a *Computer Assisted Virtual Environment*[1] (CAVE) like system is used in combination with reconstructed 3D avatars to let the remote collaborators immerse into a shared, virtual environment [60].

Teleimmersion can for example be used in remote injury assessment [48]. A therapist can treat a remote patient and demonstrate specific exercises. Forte et al. [17] use the shared virtual environment for cyberarchaeology. The collaborators can examine 3D archeological models and interact with them in real-time. Similarly, Beck et al. present an immersive group-to-group telepresence setup by using multiple Kinect sensors [1] (cf. Figure 2.2). Distributed groups of people can virtually meet. Pointing as well as tracing gestures are mutually understood in order to interact with virtual objects.

2.2 Concepts

3D-Board strives to maximize the social presence and collaboration efficiency based on three main concepts deduced from the related work.

[1] http://en.wikipedia.org/wiki/Cave_automatic_virtual_environment

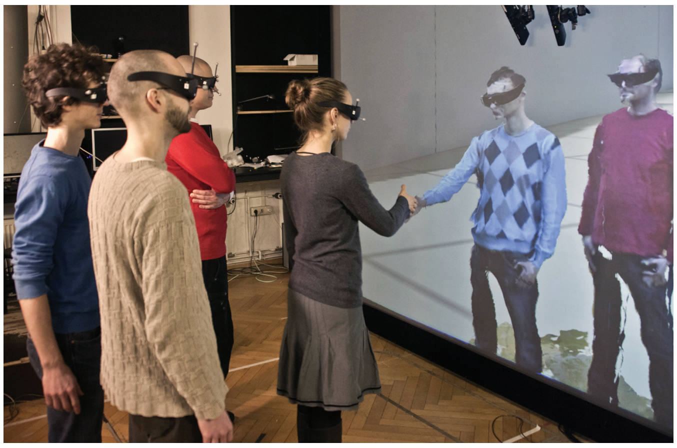


Figure 2.2: Group-to-group teleimmersion system. Image source [1].

2.2.1 Whole Body Interaction

Providing a life-sized embodiment of a remote user is paramount for large scale screens. It does not suffice to show only parts of the body, but an authentic representation is mandatory. Only then can an observer comprehend the full range of gestures of the collaborator, resulting in a raised workspace awareness [24]. At the same time, the embodiment needs to face the user. Facial expressions and gaze awareness are very important for quickly grasping the collaborator's intentions [54]. A rear view embodiment [22, 41] often leads to occlusion of deictic gestures and limits the natural interaction between the users. Thus, *3D-Board* is reconstructing a front-facing embodiment to convey deictic gestures and gaze awareness (cf. Figure 2.3).

2.2.2 Embodiment Superposition

In order to raise the social presence of the remote user, the virtual embodiment needs to be fused with the shared data [29]. The implementation of *3D-Board* is an independent application. Consequently, it is possible to superimpose the virtual embodiment on top of any content. Occlusion of important elements is avoided by making the virtual embodiment semi-transparent (cf. Figure 2.4). Thus, a face-to-face interaction with the remote collaborator is made possible and workspace awareness is raised to the level of co-located interaction.

2.2.3 Perception

As mentioned, *3D-Board* adopted the key metaphor of ClearBoard [31]. Thus, the screen should become a transparent glass window that functions as an interactive portal into the remote user's room. In order to create the

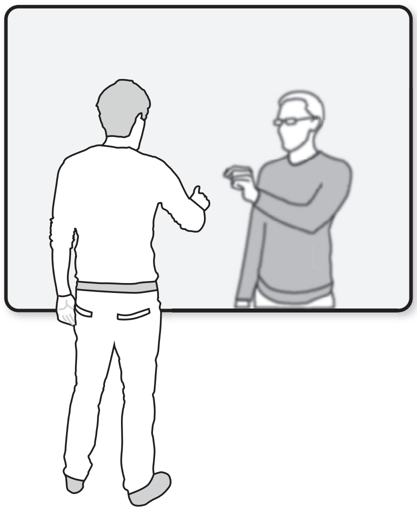


Figure 2.3: The front-facing virtual embodiment allows to perceive deixis and gaze as if the remote user would be standing behind the transparent, interactive whiteboard.

illusion that the geographically distributed person is actually standing behind the shared whiteboard requires the remote collaboration experience to be immersive. Therefore, a user should be able to perceive the remote embodiment in 3D in the same way a real person behind a glass window can be observed. The evaluation conducted by Kim et al. [37] suggests that using

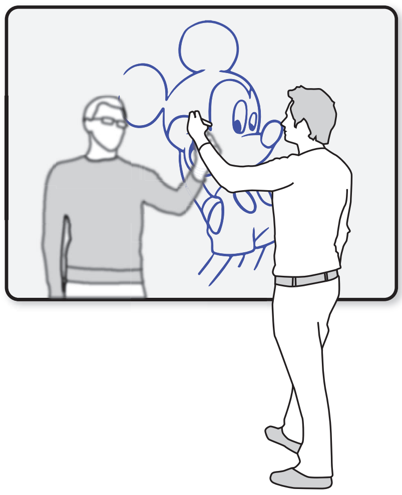


Figure 2.4: The virtual embodiment can be superimpose on top of the shared workspace to raise the social presence of the remote user.

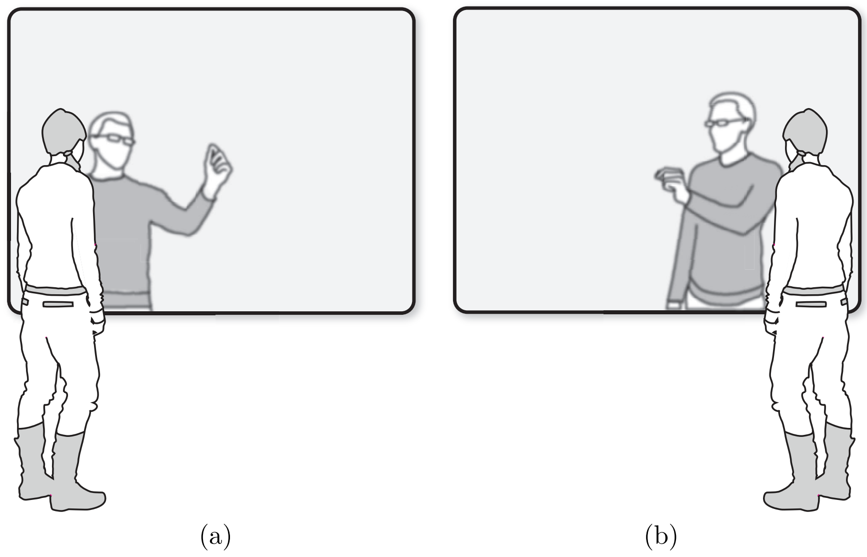


Figure 2.5: Tracking the observer's head allows to perceive the virtual embodiment in 3D while moving from left (a) to right (b).

3D instead of 2D virtual embodiments results in a significant increase in the assessment of deictic gestures.

However, an encumbrance free experience is desired. The system needs to be accurate and should not constrain the user but support the remote collaboration. Thus, *3D-Board* avoids a CAVE like system and the need for shutter glasses. Instead, the whiteboard and the cameras for 3D reconstruction are a self-contained unit. The 3D embodiment's pose and deictic gestures can be perceived by providing motion parallax via head tracking (cf. Figure 2.5).

Chapter 3

Implementation

This chapter describes the recording, visualization and perception of the virtual, front-facing 3D embodiment of a geographically distributed collaborator. The implementation of *3D-Board* is based on the requirements discussed in Section 2.2.

3.1 Setup and Hardware

With the previously discussed concepts in mind a prototypical setup was built to simulate distributed workplaces.

3.1.1 Prototype

The setup consists of two interactive whiteboards: the remote workplace for capturing the user in front of the screen and the local setup for observing the virtual embodiment. Both boards have a size of 1.6 m × 1.2 m and are operated by two Vivitek D795WT short throw projectors at a resolution of 1280 × 800 pixels. User input was captured through Anoto digital pens[1] (ADP 601) [26]. To avoid networking issues, both whiteboards are driven by the same PC with an Intel Core i7-3770, 8GB RAM and a Nvidia Quadro K2000, 2GB GDDR5.

3.1.2 Camera Placement

For capturing a remote user and visualizing the virtual embodiment three Kinect v1 depth sensors (cf. Section 1.2) are utilized (cf. Figure 5.1). All are operated at a color and depth resolution of 640 × 480 pixels at 30 frames per second.

Two cameras are attached to the top left and top right corner of the remote user's whiteboard. The depth sensors' data are used to reconstruct

[1]http://www.anoto.com/products/anoto-live-digital-pen/

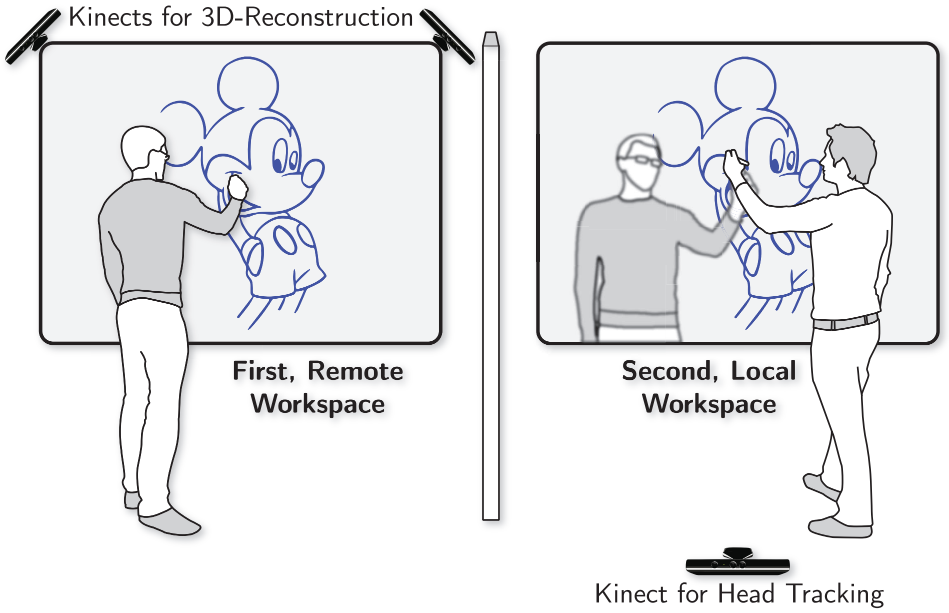


Figure 3.1: The prototype of *3D-Board* uses two Kinects to capture the interaction of the remote user with the whiteboard (left). The local user can observe the remote virtual embodiment in 3D by utilizing a third Kinect for head tracking (right).

a 3D virtual embodiment of the geographically distributed person. Rotated downwards at a 45° angle, the two depth sensors capture the user's actions from directly on the whiteboard's surface up to 1.2 m in front of the screen. Each of the sensors are capturing exactly half of the upper body of the user, with only very small overlapping regions. Both sensors are operated in near-mode (featuring valid depth data from 40 cm up to 3 m) in order to cover the whole width and height of the board. Only a minimal loss of interaction radius of about 10 cm exists in the upper corner regions of the whiteboard. Since the two Kinects are rotated inwards to track the user, their output also appears tilted (cf. Figure 3.2). In order to reconstruct a virtual 3D embodiment of the user out of the two rotated images, the cameras need to be calibrated. Thereby, the output of both Kinects can be transformed into a common coordinate space. This will fuse the output and enable rendering the scene from a single, virtual viewpoint.

This virtual viewpoint is determined by the third Kinect. At the second, local workspace, this sensor is capturing the full size of the whiteboard from the back of the room. The camera tracks the head of the local observer to support motion parallax by changing the viewpoint of the rendering accordingly.

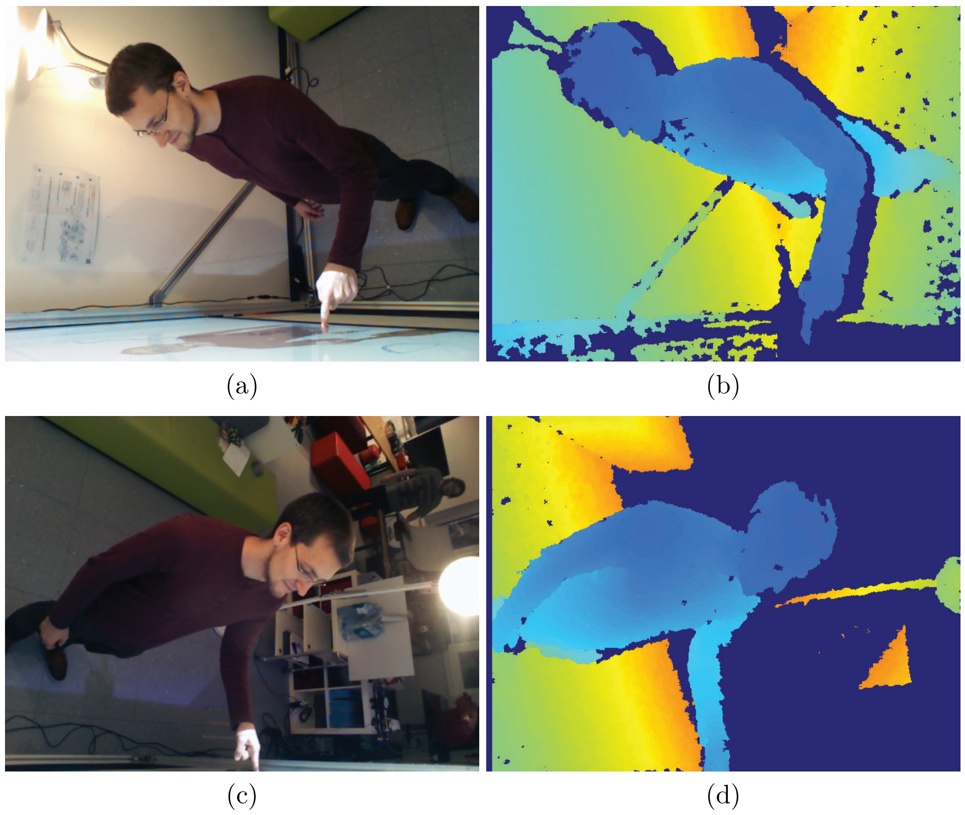


Figure 3.2: The color and depth frame of the left sensor are shown in (a) and (b) and the frames of the right Kinect in (c) and (d). Placed in the upper corners of the whiteboard and rotated inwards, both cameras capture half of the upper body of the user, with only very little overlap. Valid depth values range from 40 cm up to 3 m.

3.2 Camera Calibration

As briefly mentioned, calibrating the two Kinect sensors, is essential for reconstructing the virtual embodiment. Otherwise rendering the recorded depth data and merging the output of multiple Kinects into a single image would not be possible. The camera calibration computes the intrinsic (focal length, principal point, distortion coefficients) and extrinsic (position and pose of the camera) parameters. Thus, before explaining the calibration procedure the camera's parameters are discussed.

3.2.1 Extrinsic Camera Parameters

The extrinsic parameters consist of the translation vector $\vec{t}$ and rotation matrix R

$$\vec{t} = \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} \qquad R = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}. \tag{3.1}$$

This transformation describes the pose of a calibration pattern, as it is used during the calibration procedure (discussed in Sec. 3.2.5), relative to the camera [6, p. 394]. Paraphrased, the calibration pattern defines a world coordinate system that is viewed from the two camera coordinate systems centered at the Kinect sensors. By applying the extrinsic rigid body transformation, the point $P_w = \begin{pmatrix} x_w, & y_w, & z_w \end{pmatrix}$ of an object, such as the user in front of the screen, is transformed from the world coordinate system to the camera coordinate system, resulting in the point $P_c = \begin{pmatrix} x_c, & y_c, & z_c \end{pmatrix}$:

$$P_c = RP_w + \vec{t}. \tag{3.2}$$

Transforming from world to camera space, as depicted in Figure 3.3, can also be written in matrix notation. Therefore, homogeneous coordinates are used to combine the translation matrix T with the rotation matrix R to form a single extrinsic transformation matrix M_e:

$$P_c = T \cdot R \cdot P_w = M_e \cdot P_w = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_w \\ y_w \\ z_w \\ 1 \end{pmatrix} = \begin{pmatrix} x_c \\ y_c \\ z_c \\ 1 \end{pmatrix}. \tag{3.3}$$

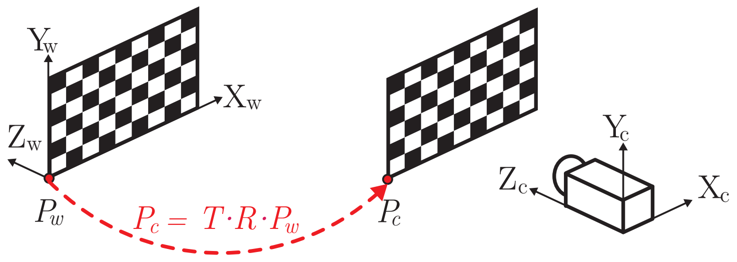


Figure 3.3: The pose of the checkerboard calibration target relative to the camera is described by the rotation R and translation T. Applying this transformation to a point P_w in the world coordinate system transforms it to the camera coordinate system, resulting in P_c.

3.2.2 Intrinsic Camera Parameters

There are two important intrinsic camera parameters, the focal length and the principal point.

Focal Length

Assuming an ideal pinhole camera model with a frontal image plane (cf. Figure 3.4), the focal length f is the distance of the image plane (image sensor) to the optical center O. The elements of an image sensor are not necessarily square but can be rectangular. Thus, the focal length is actually the product of the physical focal length and the size s_x and s_y of the elements:

$$f_x = f \cdot s_x, \qquad f_y = f \cdot s_y. \tag{3.4}$$

Principal Point

The principal point c is the point at the intersection of the optical axis with the image plane. It is very likely that the image sensor of a camera is

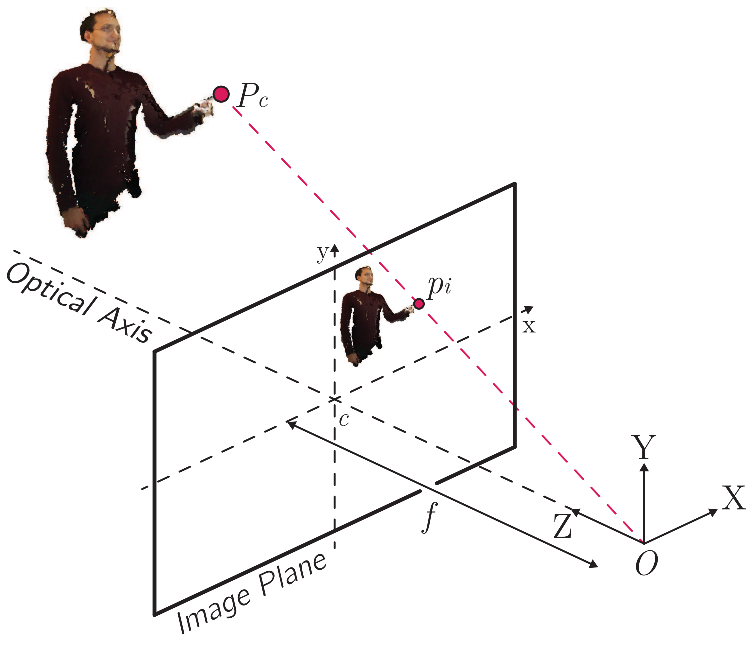


Figure 3.4: An ideal pinhole camera model with a frontal image plane.

displaced from the optical axis by c_x and c_y, resulting in the principal point

$$c = \begin{pmatrix} c_x \\ c_y \\ f \end{pmatrix}. \tag{3.5}$$

Therefore, a point $p_i = \begin{pmatrix} x_i, & y_i \end{pmatrix}$ on the image plane with continuous coordinates is mapped to a point p'_i on the image sensor with actual pixel coordinates[2] u and v by adding the displacement:

$$p'_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix} + \begin{pmatrix} c_x \\ c_y \end{pmatrix} = \begin{pmatrix} u \\ v \end{pmatrix}. \tag{3.6}$$

Projecting from camera to image space

As the 3D point P_c of an object in camera space[3] is imaged by the Kinect sensor, it is projected onto the image plane, resulting in the 2D point p'_i. The relation between p'_i and P_c is given via similar triangles (cf. Figure 3.4):

$$\frac{x_c}{z_c} = \frac{x_i}{f_x}, \qquad \frac{y_c}{z_c} = \frac{y_i}{f_y}. \tag{3.7}$$

Perspectively mapping the 3D point P_c to the 2D point p_i is therefore described as

$$u = f_x \frac{x_c}{z_c} + c_x, \qquad v = f_y \frac{y_c}{z_c} + c_y. \tag{3.8}$$

In order to easily process this nonlinear, perspective transformation as a linear matrix-vector multiplication, homogeneous coordinates are used. At first, a projection matrix M_p can be constructed [45, pp. 52–53] by combining the canonical projection matrix M_0 and the camera intrinsics matrix M_i that is composed out of the focal length (3.4) and principal point (3.5) as

$$M_p = M_i \cdot M_0 = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{3.9}$$

Then, by utilizing homogeneous coordinates and the projection matrix M_p, a point in world space is projected into image space:

$$p'_i \cdot z_c = M_p \cdot P_c = \begin{pmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x_c \\ y_c \\ z_c \\ 1 \end{pmatrix} = \begin{pmatrix} u \cdot z_c \\ v \cdot z_c \\ z_c \end{pmatrix} = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \cdot z_c. \tag{3.10}$$

[2]As will be discussed in section 3.2.6, p'_i is a pixel on the depth texture received from the Kinect sensor.

[3]Leaving the extrinsic transformation aside, the camera space is equal to the world space originating at the camera's optical center.

This projection can also be described as shifting the imaged pixel back along the light ray. The point p_i' and the center of the camera O describe the vector $\vec{q} = \begin{pmatrix} u, & v, & 1 \end{pmatrix}^T$. Scaling the point along the vector by the factor of z_c results in P_c, the point in camera space[4], as depicted in Figure 3.4.

3.2.3 Projective Transformation

The complete projective transformation comprises of the intrinsic projection matrix M_p (3.9) and the extrinsic transformation matrix M_e (3.3). It can be used to transform a world coordinate point P_w to a point p_i in image space

$$p_i' \cdot z_c = \begin{pmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_w \\ y_w \\ z_w \\ 1 \end{pmatrix} = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \cdot z_c. \quad (3.11)$$

3.2.4 Lens Distortion

So far, the discussed camera was based on an ideal pinhole model. However, the lenses of real cameras introduce additional distortion coefficients [33, Sec. 12.9]:

1. **Radial distortion** occurs when light rays farther away from the center of the lens are bent more than rays close to the center due to poor quality lenses.
2. **Tangential distortion** is the result of the lens being decentered from the optical axis.

The skewedness or diagonal distortion of the image sensor is insignificant in most case and therefore not discussed. The radial distortion parameters k_1, k_2, k_3 and tangential distortion coefficients p_1, p_2 can be modeled as polynomials [6, p. 375–376]. Since the lens of the camera is placed in front of the image sensor, the distortion takes effect after the projection. Thus, the distortion parameters are applied after the projection matrix M_0, but before the image to sensor mapping, using the intrinsics matrix M_i of equation (3.9). For the sake of brevity, the more complex camera model including the lens distortion is not explained in depth, but the interested reader is referred to [33, p. 341–357] and [6, p. 375–378].

3.2.5 The Calibration Procedure

Since the cameras are static, computing the intrinsic and extrinsic parameters of the camera is an offline process that needs to be done once before

[4]Since p_i' is equal to a pixel on the Kinect's depth texture, the value of this pixel is the depth z_c of the point in 3D space (cf. Sec. 3.2.6).

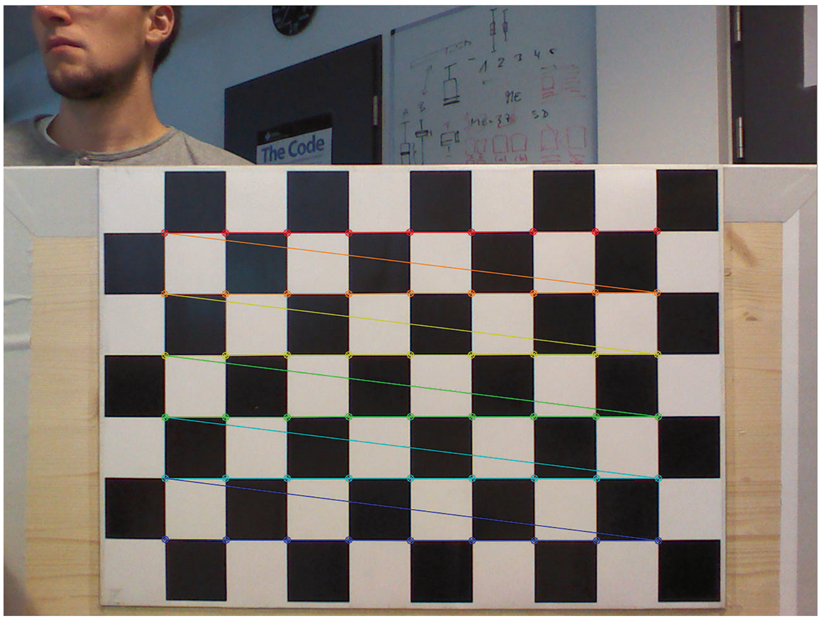


Figure 3.5: The detected corners of the checkerboard.

the scene can be rendered. The calibration routines are implemented in C++ using *OpenCV 2.4.8*[5] [6, p. 389] and are based on Zhang [64] and Brown [7].

Computing the Calibration Parameters

In order to compute the previously discussed intrinsic and extrinsic parameters, multiple images of a checkerboard (9×6 inner corners of the black and white squares with a side length of 59 mm) calibration target in different poses need to be taken. The corner points of the checkerboard in each image can be identified by calling *OpenCV's* `findChessboardCorners` method (cf. Figure 3.5).

An identified corner point P_w in world coordinates is perspecitvely transformed onto the image sensor, resulting in p_i', as previously described (cf. Equation 3.11). Thus, the linear mapping from a plane in 3D space onto a plane in 2D space, results in a planar homography [6, p. 384–387]. The homography matrix H is the combination of the intrinsic projection M_p (3.10) and the extrinsic transformation M_e (3.3), that is

$$H = M_p \cdot M_e. \tag{3.12}$$

Thus, H is holding the intrinsic and extrinsic parameters and can be computed based on multiple 2D-3D point correspondences of the form

$$p_i' = sHP_w. \tag{3.13}$$

[5]http://opencv.org/

The arbitrary scale factor s is unknown and can be ignored since only the ratio of the elements of the homogeneous matrix H is of importance [27, p. 33]. In addition, the points on the calibration target can be considered as lying in one plane at depth $z_w = 0$. This further reduces the homography to a 3×3 matrix if the rotation matrix is broken up into three columns $R = \begin{pmatrix} r_1, & r_2, & r_3 \end{pmatrix}$ [6, p. 386]. Therefore, p'_i can be computed as

$$\begin{aligned} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} &= s \cdot M_p \cdot \begin{pmatrix} r_1, & r_2, & r_3, & \vec{t} \end{pmatrix} \cdot \begin{pmatrix} x_w \\ y_w \\ 0 \\ 1 \end{pmatrix} \\ &= s \cdot M_p \cdot \begin{pmatrix} r_1, & r_2, & \vec{t} \end{pmatrix} \cdot \begin{pmatrix} x_w \\ y_w \\ 1 \end{pmatrix} . \end{aligned} \tag{3.14}$$

From this homography the unknown parameters can be solved by deriving a system of linear equations of the form $A \cdot b = 0$ [6, p. 389–392]. A is holding the equations of the known homographies and b is the vector of the unknown intrinsic and extrinsic parameters. In order to determine the lens distortion parameters, the calibration parameters computed without distortion are used to numerically solve a larger system of equations [6, p. 391–392]. The parameters are then refined again for each image of the calibration target.

The homography of a single image yields eight parameters when mapping the four corner points of the plane of the calibration pattern to the u and v coordinates of the image plane. The five distortion coefficients depend only on the mapped 2D points and can thus theoretically be solved using one image. The four intrinsic parameters are bound to the six extrinsics that are unique for each view. Therefore, at least two images of the calibration pattern are needed to solve all unknowns. However, due to noise and numerical stability, usually more than 20 images proved to be needed per camera for an accurate measurement of all parameters [6, p. 388].

Procedure

The calibration procedure to compute the intrinsic, distortion and extrinsic parameter comprises of the following steps:

1. For higher precision, the Kinect's RGB sensor is set to take pictures at a resolution of 1280×960 with 15 frames per second. Since the resolution used for rendering is set to 640×480, the obtained intrinsic parameters need to be divided in half accordingly. However, the extrinsic parameters as well as the distortion coefficients do not scale with an increased resolution.

2. For each of the two Kinect sensors in the upper left and upper right corners of the whiteboard, more than 20 images of different poses of the checkerboard calibration target are taken separately for each sensor.
3. The corner points are computed at the time of each taken image and can then be used for *OpenCV's* `calibrateCamera` routine. This method computes the intrinsic, extrinsic and distortion parameters for each of the images by using the planar homography.
4. For the last picture, the calibration target is imaged from both Kinect cameras simultaneously (cf. Figure 3.5), in order to be able to compute the extrinsics for both cameras for the same calibration target. The rotation and transformation between the two cameras could also be computed using stereo camera calibration [6, p. 427–430].

3.2.6 Using Camera Calibration for 3D Reconstruction

Once finished with the calibration, the computed coefficients can be used for rendering the 3D virtual embodiment imaged by the two Kinect cameras.

The projective transformation (cf. Equation 3.11) was used to project the imaged world coordinate point P_w onto the image sensor, resulting in the 2D point p_i' scaled by z_c. In order to reconstruct an imaged object, this transformation needs to be reversed by first projecting the Kinects' image data into 3D space and then transforming the output of both Kinects into a common coordinate system.

Undistorting the camera image

Before reconstructing the scene the lens distortion needs to be removed. Undistorting an image requires the computation of an undistortion map by calling *OpenCV's* `initUndistortRectifyMap` function [6, p. 396–397]. Based on the distortion coefficients the map relates an pixel in the image to the mapped, undistorted point. Once the map is computed for the sensor, it can be used to quickly undistort every image using `remap`. The Kinect sensors seem to have high quality lenses and thus undistorting an image has only very little influence (cf. Figure 3.6).

Projecting from image to camera space

The point p_i' (3.6) represents a pixel of a Kinect's depth texture. The value of the pixel stored in the texture is equal to the corresponding depth z_c of the imaged point in 3D space. In analogy to (3.8), the image space point p_i' can be projected to the 3D point P_c in camera space:

$$x_c = z_c \frac{x_i}{f_x} = z_c \frac{u - c_x}{f_x}, \qquad y_c = z_c \frac{y_i}{f_y} = z_c \frac{v - c_y}{f_y}. \tag{3.15}$$

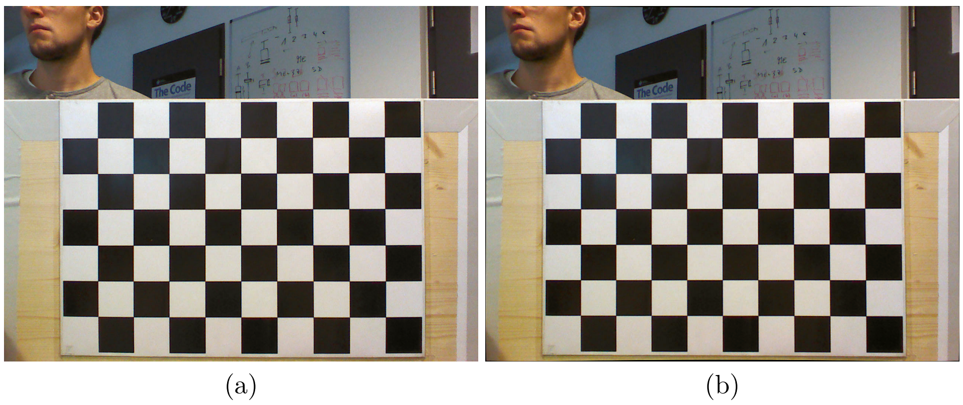


Figure 3.6: Due to the high quality lenses of the Kinect the undistortion of the color image (a) has almost no effect (b).

This is equal to the left inverse [28, p. 79] of the projection matrix M_p (3.9) to invert Equation (3.10):

$$P_c = M_p^{-1} \cdot p_i' \cdot z_c = \begin{pmatrix} \frac{1}{f_x} & 0 & -\frac{c_x}{f_x} \\ 0 & \frac{1}{f_y} & -\frac{c_y}{f_x} \\ 0 & 0 & 1 \\ 0 & 0 & \frac{1}{z_c} \end{pmatrix} \cdot \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \cdot z_c = \begin{pmatrix} x_c \\ y_c \\ z_c \\ 1 \end{pmatrix}. \qquad (3.16)$$

Hence, a depth reading acquired from a Kinect sensor will be projected into 3D space from the camera's point of view, resulting in a point P_c in the camera coordinate space. In this way, a cloud of 3D points is produced using all the depth readings of the Kinect (as will be explained in detail in Section 3.3.2). Hence, a point P_w on the object imaged by both cameras results in a point P_{c_1} in the first camera coordinate system and a point P_{c_2} in the second camera coordinate system (cf. Figure 3.7(a)). Consequential, the point clouds of both Kinects are defined in two different coordinate systems and are thus misaligned (cf. Figure 3.7(b)).

Transforming from the Camera to the World Coordinate System

In order to reconstruct the object, the point clouds need to be transformed into the shared world coordinate system originating at the calibration pattern imaged by both Kinect sensors. This can be done by applying the inverse of the extrinsic transformation (3.2) to each point P_c of the point cloud in the camera coordinate system (cf. Figure 3.7(c)). Since rotation matrices are orthonormal, the inverse of R is its transposed R^T [49, p. 96], resulting in the equation:

$$P_w = R^T P_c - R^T \vec{t}. \qquad (3.17)$$

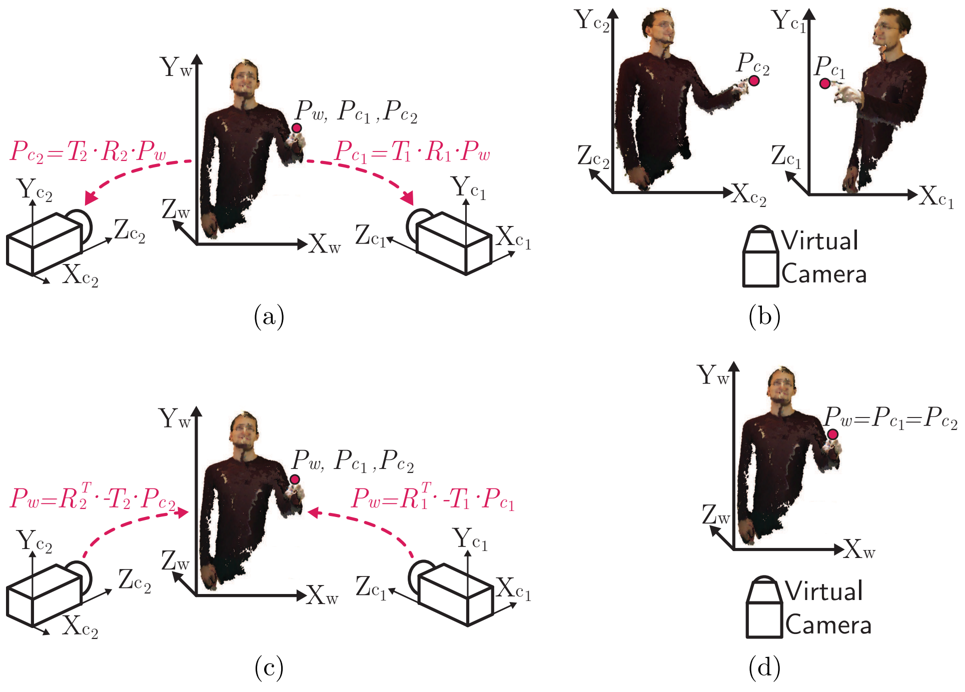


Figure 3.7: An object imaged by both Kinects (a) results in two reconstructed embodiments in two different camera coordinate systems, if visualized using a virtual camera (b). Applying the inverse of the extrinsic transformation (c) aligns both point clouds in the same coordinate system and merges the two images (d).

Applying the inverse to each point cloud will transform the points P_{c_1} of the first camera and the points P_{c_2} of the second camera into the common world coordinate system. Thus, both renderings of the same object will align (cf. Figure 3.7(d)).

Coordinate systems in *OpenCV* and *DirectX*

The transformation acquired from *OpenCV* will be used to transform the point clouds in *DirectX*[6], as will be discussed in the visualization Section 3.3.2. Since *DirectX* uses a left handed coordinate system and *OpenCV* an image coordinate system (cf. Figure 3.8), the mirrored y-axis needs to be accounted for.

Thus, to convert the extrinsic transformation obtained from the camera calibration from *OpenCV* to *DirectX* the second element t_y of the translation vector needs to become its additive inverse: $\vec{t} = \begin{pmatrix} t_x, & -t_y, & t_z \end{pmatrix}^T$. For the rotation, *OpenCV's* calibration routine returns a 3-tuple vector

[6]http://en.wikipedia.org/wiki/DirectX

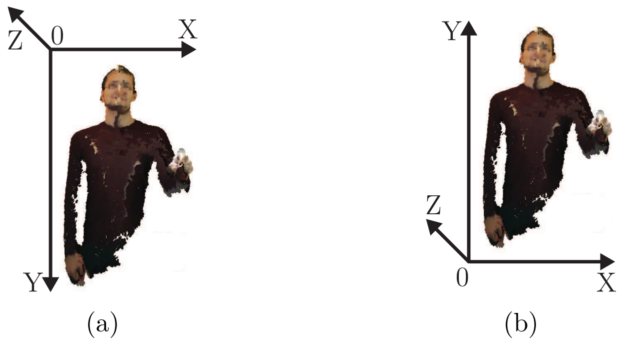


Figure 3.8: The difference between the image coordinate system of OpenCV (a) and the left handed coordinate system of DirectX (b) is the flipped y-axis.

$\vec{r} = \begin{pmatrix} r_x, & r_y, & r_z \end{pmatrix}^T$ describing an axis-angle rotation [61, Ch. 5.4]. This three dimensional vector normalized to unit length defines the axis of rotation, while the length of the vector is equal to the magnitude of the angle about this axis. Hence, to invert only the y-coefficient of the rotation it suffices to invert the second element of the rotation vector. This vector can then be transformed into the rotation matrix R by using the *Rodrigues* transform [6, p. 401–403] for building the extrinsic transformation Equation (3.17). Similarly, the additive inverse of f_y, the intrinsic focal length in y, needs to be built.

Inaccuracy and Manual Refinement

Although *OpenCV's* camera calibration routines feature sub-pixel accuracy, the point clouds of both Kinect cameras are still misplaced by a few centimeters (cf. Figure 3.9(b)) after the calibration. The author assumes that the observed misalignment between the two point clouds has multiple causes:

1. The official Kinect SDK is used to map the Kinect's depth texture into the same space of the color texture, as will be discussed in the visualization section 3.3.1. The method uses factory calibration parameters that probably do not match the intrinsics of the used Kinect models. For lack of time the mapping was not implemented with the accurately computed camera parameters (as will be described in Section 3.3.1).
2. Maimone et al. [47] evaluated that the Kinect's depth resolution falls off quadratically with increased distance. At a distance of 1500 mm (the common operational distance of *3D-Board*) the measured distances vary by 30 mm on average. Also Kainz et al. [36] witnessed inaccuracies between the depth readings of multiple Kinects. The misalignment was overcome by triangulating a calibration object and fitting a polynomial function to the depth errors.

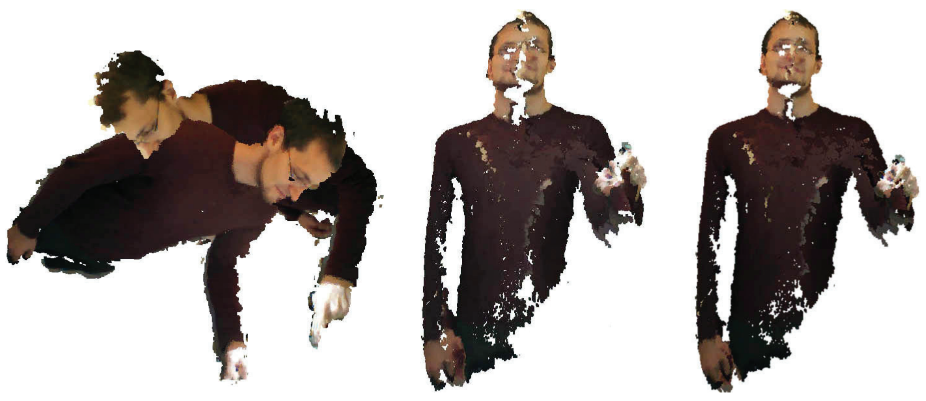

(a) (b) (c)

Figure 3.9: The reprojected point clouds of both Kinects without any calibration (a), with Zhang's calibration (b) and additional manual fitting (c).

3. Due to the placement of the cameras, the calibration target covers only a small viewing area when imaged from both Kinects for extrinsic calibration. This could lead to a misalignment for the larger portion of the Kinect's view.

In order to correct the reprojection error two approaches were tested:

1. The *Iterative Closest Point* (ICP) algorithm [51, 62] tries to minimize the distance between a reference and a source point cloud by iteratively revising the transformation between the corresponding points. A variant of this algorithm implemented in *Meshlab 1.3.3*[7] was tested. The reprojected point clouds of both Kinects were saved to a file and reloaded in *Meshlab* in order to utilize the ICP implementation. However, registering the point clouds is only successful if their overlap is big enough. Due to the large distance between the Kinect camera's this is almost never the case.
2. A more promising but likewise tedious approach is to manually refine the extrinsic transformation of both point clouds by adding additional transformation matrices until the output visually aligns (cf. Figure 3.9(c)).

3.3 Visualization

This section describes the visualization of the virtual embodiments using the obtained camera parameters. For programming the Kinects the official Kinect SDK[8] was used. The rendering of the 3D scene is implemented in

[7]http://meshlab.sourceforge.net/
[8]http://www.microsoft.com/en-us/kinectforwindowsdev/

C++, *DirectX 11* and the *HLSL* shading language. Image processing techniques are realized using *OpenCV 2.4.8*. The final output is rendered into a *Windows Presentation Foundation*[9](*WPF*) window using C#.

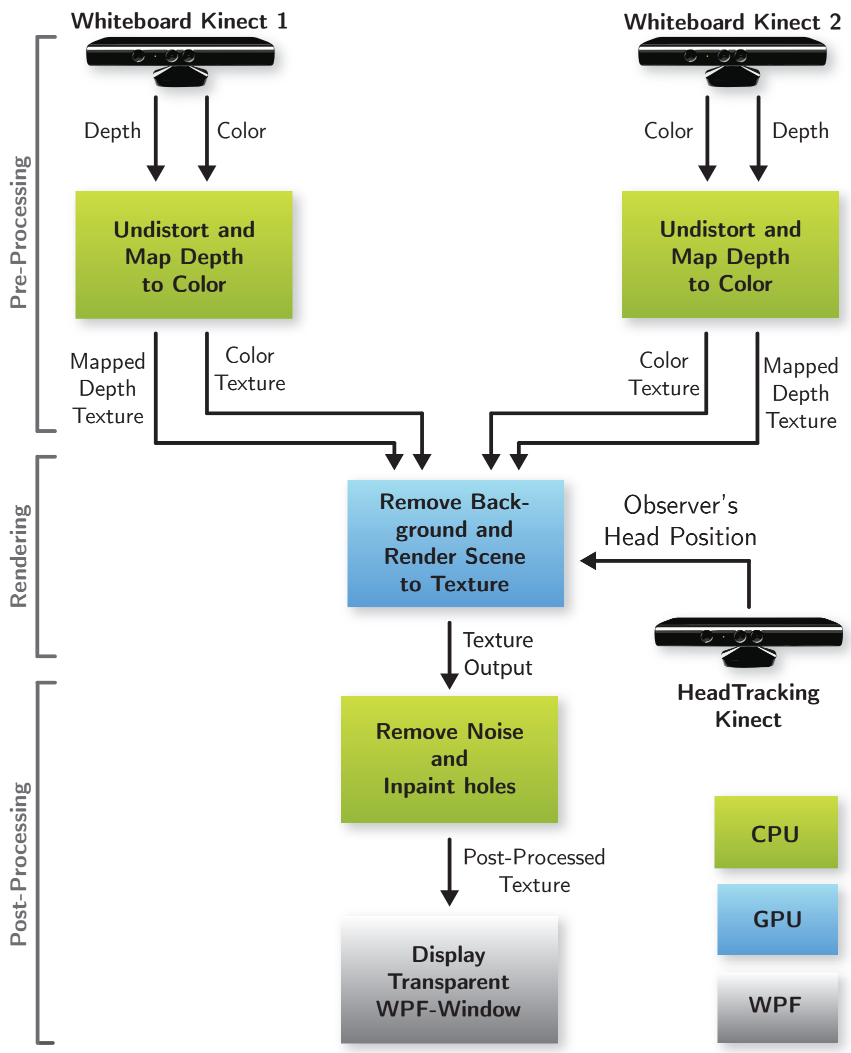


Figure 3.10: The data processing and rendering pipeline of *3D-Board*.

The implemented data processing and rendering pipeline (cf. Figure 3.10) is composed of a pre-processing, rendering and post-processing stage:

1. Pre-Processing: the pre-processing stage prepares the color and depth data for rendering on the GPU.
2. Rendering: the processed color and depth textures are handed over to the GPU. The geometry shader creates the point clouds out of the

[9]http://msdn.microsoft.com/de-de/library/ms754130(v=vs.110).aspx

depth data and transforms it into a common coordinate system to merge the visual output of both cameras.

3. Post-Processing: the image rendered by the GPU is copied to system memory for further post-processing on the CPU. Holes are filled before displaying the processed output in a WPF window.

3.3.1 Pre-Processing

The color and depth frames need to be pre-processed in preparation for the rendering stage. Part of this stage is to undistort each frame obtained from a Kinect sensor, as previously described in Section 3.2.6. However, besides the undistortion further processing needs to be executed.

Depth to Color Registration

Since the RGB camera and the IR sensor of a Kinect are placed apart from each other by a few centimeters (cf. Section 1.2), the color and depth data are not received from the same viewport and thus need to be registered. Because the extrinsic camera parameters were obtained from the color camera, the depth frame needs to be mapped onto the RGB frame. This registration can be done using the `MapDepthFrameToColorFrame` function of the official Kinect SDK. The function outputs the position of the depth pixels mapped on the color frame. Thus, the depth map needs to be filled with the depth values of the corresponding mapped positions (cf. Listing 3.1).

Algorithm 3.1: Algorithm for mapping depth to the color.

```
    Compute new values for depth map $D$ after mapping to the color frame.
1:  MapDepthToColor($D$)
                                      ▷ Compute mapped depth positions $M$
2:      $M \leftarrow$ MapDepthFrameToColorFrame($D$)
3:
4:    ▷ Compute depth value for the corresponding mapped depth position
5:      $C \leftarrow D$                          ▷ Copy the original depth map values
6:      for $i \leftarrow 0$, $i < D_{count}$ do
7:          $x, y \leftarrow$ GetPos($M_i$)
8:          $j \leftarrow x + y \cdot D_{width}$
9:          if $x \geq 0 \wedge x < D_{width} \wedge y \geq 0 \wedge y < D_{height}$ then
10:             $D_i = C_j$
11:         else
12:             $D_i = 0$
13:         end if
14:     end for
15: end
```

Since `MapDepthFrameToColorFrame` uses predefined values for the registration that do not fit perfectly for each model of the Kinect, it lacks accuracy (as previously discussed in Section 3.2.6). A more precise solution, that was not performed, would have been to map the depth values based on the camera calibration parameters:

1. At first, stereo camera calibration [6, p. 427–430] needs to be performed with both the RGB and the depth camera. This results in the intrinsic matrix M_{i_d} of the depth camera and the intrinsic matrix M_{i_c} of the color camera as well as the rotation matrix R and the translation vector $\boldsymbol{t}$ describing the relative transformation between the two sensors.
2. Next, each pixel of the depth texture p'_{i_d} has to be projected into 3D space using the corresponding depth value z_i stored in the texture and the inverse of M_{i_d} (cf. Equation (3.16)):
$$P_c = M_{i_d}^{-1} \cdot p'_{i_d} \cdot z_i. \tag{3.18}$$
3. The resulting 3D point P_{c_d} in the camera space of the depth sensor is transformed to a 3D point P_{c_c} in the camera space of the color sensor by applying the extrinsic parameters of the stereo calibration (cf. Equation (3.2)):
$$P_{c_c} = R \cdot P_{c_d} + \boldsymbol{t}. \tag{3.19}$$
4. Once the point is transformed to the color camera space, it can be reprojected onto the RGB sensor, resulting in the color pixel p'_{i_c}, by applying M_{i_c} (cf. Equation (3.10)):
$$p'_{i_c} = \frac{M_{i_c} \cdot P_{c_c}}{z_i}. \tag{3.20}$$

Color Matching

The Kinect's color camera has white balancing and exposure time enabled by default. Thus, to obtain consistent RGB values across all sensors, the white balancing is set to a fixed value of 5200 kelvin and auto exposure is disabled for every camera via the SDK's `ColorCameraSettings`.

For the presented prototype the produced results were deemed good enough and no further enhancements were implemented. However, to improve the quality, all RGB cameras should be calibrated towards a physical color checker target [30] or even match the corresponding colors in each frame based on the depth data [46].

Depth Map Filtering

The Kinect's depth data show a considerable amount of noise. Especially for an increased depth range and at the region of interference of both Kinect

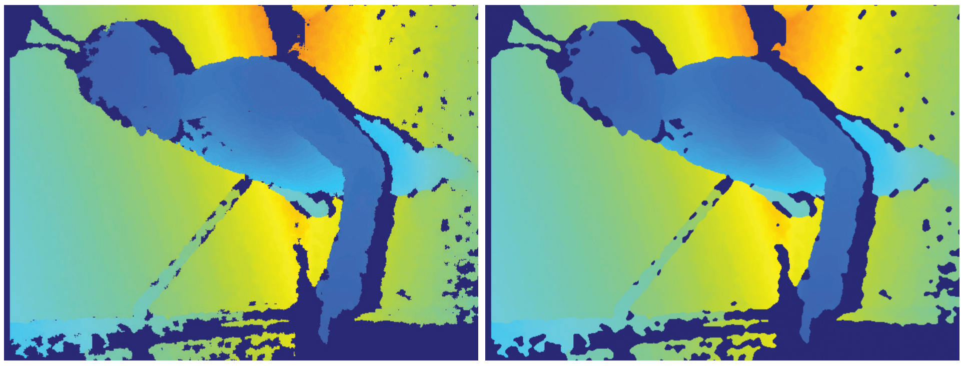

(a) (b)

Figure 3.11: The median filter with the kernel size of 3×3 pixels smooths the depth map and closes small holes while preserving the edges.

cameras missing depth readings occur. In order to smooth the depth map and fill small holes, a median filter is applied (cf. Figure 3.11). It replaces the pixels at position (u, v) of the depth map D_i with the median of all pixels in the filter region R [9, Ch. 5.4.2]:

$$D_i(u, v) \leftarrow \text{median}\left\{I(u+i, v+j) | (i, j) \in R\right\}. \tag{3.21}$$

The advantage of a median filter is its edge preserving capability. However, the edge preservation is not pixel exact and can thus lead to extended edges that consequently pick up wrong color values from the background. Although the effect is very small, more advanced methods, such as a bilateral filter [8, Ch. 5.2] that conserves border regions in conjunction with the color texture [12, 40], could improve the overall quality.

The median filter was implemented using *OpenCV's* `medianBlur` function. For the kernel size a relatively small number of 3×3 pixels was chosen, since the *OpenCV* implementation is rather costly and has to be executed for every utilized depth map.

3.3.2 Point Cloud Rendering

Both, the pre-processed color and depth textures are used for rendering a colored cloud of 3D points. The point cloud consists of fixed-size quads created for each pixel of the depth texture in the geometry shader. The geometry shader is based on a modified Kinect for Windows Sample Program[10] released under the Apache License[11]. The data flow between the program and the shaders for rendering a Kinect's output is depicted in Figure 3.12.

[10]http://kinectforwindows.codeplex.com/

[11]http://www.apache.org/licenses/LICENSE-2.0.html

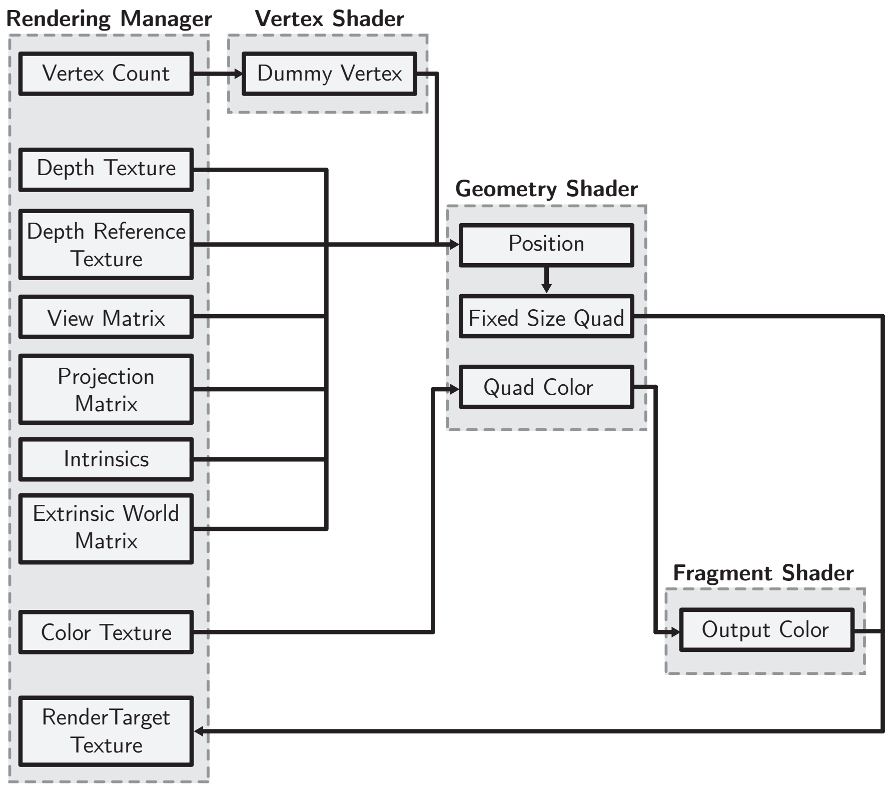


Figure 3.12: The data flow between the program and the shaders to render a point cloud into a texture that can be used for post-processing.

Initialization

Distilled, the initialization of rendering a Kinect's depth and color output comprises of the following steps:

1. Rendering a 3D point for each depth pixel requires the primitive topology to be set to a `POINTLIST`.
2. The color and depth as well as a depth reference texture are set as a resource for the geometry shader. The depth reference frame is later on used for removing the background of the scene.
3. Since all vertices are created in the geometry shader, the vertex buffer provided for the shader pipeline is empty. In order to trigger the vertex shader without providing any vertices, the `Draw(vertexCount, startLocation)` call needs to be executed with `vertexCount` set to the resolution of the depth imager to render a vertex for each pixel of the depth texture.

The following listing depicts the initialization of the *DirectX* context and the rendering pipeline:

```
void InitDirectXDevice()
{
  // create empty vertex buffer
  D3D11_SUBRESOURCE_DATA* vertices = NULL
  dxDevice->CreateBuffer(&bufferDescription, vertices, &vertexBuffer);

  // set primitive topology for rendering to a pointlist
  dxContext->IASetPrimitiveTopology(D3D11_PRIMITIVE_TOPOLOGY_POINTLIST);
  ...
}

void RenderScene()
{
  for (auto kinect : sensors)
  {
    ...
    // Init the textures for the geometry shader
    dxContext->GSSetShaderResources(0, 1, kinect->depthTex);
    dxContext->GSSetShaderResources(1, 1, kinect->depthReferenceTex);
    dxContext->GSSetShaderResources(2, 1, kinect->colorTex);

    // Draw the point cloud
    dxContext->Draw(kinect->depthPixelCount, 0);
  }
  ...
}
```

3D Point Acquisition

The 3D points for rendering the point cloud are acquired from the depth texture in the geometry shader (the complete source code can be found in the Appendix A.2). Hence, the vertex shader does nothing but calling the geometry shader by passing an empty structure:

```
struct GeoShaderInput { };

GeoShaderInput VertexShader()
{
    return (GeoShaderInput)0;
}
```

Therefore, the geometry shader is executed as often as there are pixels in the depth texture, specified by `depthPixelCount` in the `Draw()` call. Consequently, the subsequent `PrimitiveID` is equal to the index of a pixel. By using this index, the depth value of the corresponding pixel can be read from the depth texture as well as from the depth reference texture used for background subtraction:

```
Texture2D<int> depthTex   : register(t0);
Texture2D<int> depthRefTex   : register(t0);
static const int depthWidth = 640;
static const int depthHeight = 480;
```

```

void GeometryShader(..., uint primID : SV_PrimitiveID)
{
  // sample depth from texture using the current pixel index primID
  int3 texCoord = int3(primID % depthWidth, primID / depthWidth, 0);
  // depth from texture in millimeters - converted to meters
  float depth = depthTex.Load(texCoord) / 1000.0f;
  float depthRef = depthRefTex.Load(texCoord) / 1000.0f;

  // background subtraction
  if (invalidPixel(depth, depthRef)) { return; }
  ...
}
```

Background Removal

Since only the virtual representation of the user is of importance, the background scene is removed in the geometry shader. To this end, a reference depth frame D_r is prerecorded for simple frame differencing. This reference frame shows the background scene without any foreground objects (e.g. the user), as depicted in Figure 3.13 (a). A pixel at position (u, v) is considered being part of a valid foreground object if the difference between the reference frame and the current depth frame D_i (cf. Figure 3.13 (b)) is larger than a threshold τ, empirically chosen as 0.5 m. Otherwise it is deemed to be a background pixel and thus discarded as in

$$D_i(u,v) = \begin{cases} D_i(u,v) & \text{for } |D_i(u,v) - D_r(u,v)| > \tau, \\ 0 & \text{otherwise.} \end{cases} \tag{3.22}$$

Therefore, only the foreground pixels (cf. Figure 3.13 (c)) for every depth texture $i = 1, ..., n$ remain for further processing.

The skeleton data provided by the Kinect could also have been used to segment the user from the background. However, a Kinect sensor can only

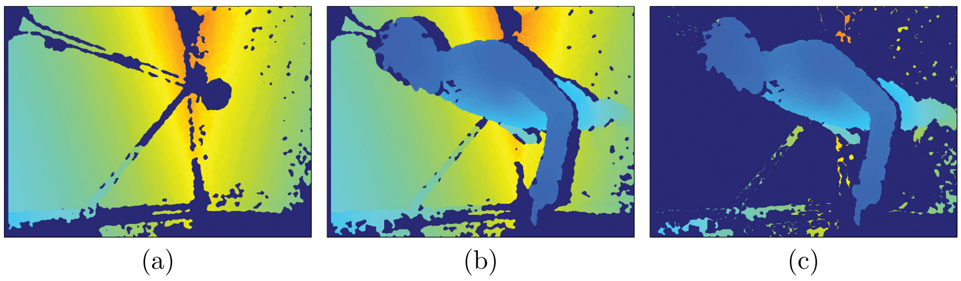


Figure 3.13: Out of a depth reference frame (a) and the current depth frame (b) the scene is reconstructed showing only the foreground objects (c). Most of the remaining noise will not be visible in the final rendering since it is outside of the observable scene.

compute the skeleton data in horizontal orientation. All cameras used for reconstruction are tilted. Therefore, the Kinect SDK's background removal capabilities cannot be utilized.

Scene Reconstruction

After removing the background, the 3D points, acquired from the depth texture, are used to reconstruct the foreground objects. This is done by projecting the pixels on the imager into camera space, as described previously in Section 3.2.6. Each pixel of the depth map is equal to a point p'_i (cf. Equation (3.6)) on the image plane with the coordinates (u, v)[12]. The value stored for each pixel is the corresponding depth value z_i of the point P_c (cf. Equation (3.2)) in 3D camera space. Thus, a depth pixel in image space can be projected into camera space by using the inverse of the intrinsic matrix, as previously shown in Equation (3.16).

Next, the resulting point cloud is transformed to world space to merge the output of both Kinects in a common coordinate system (cf. Section 3.2.6). As shown in Equation (3.17), P_c is transformed to a point in world coordinates P_w by applying the inverse of the extrinsic transformation:

```
cbuffer constantBuffer : register(b0)
{
  matrix instrinsicsInverse;
  matrix extrinsicsInverse;
  ...
};

void GeometryShader(...)
{
  // read depth from texture
  ...
  // project from image to camera space using the intrinsic matrix
  float4 imagePos = float4(texCoord.xy * depth, depth, 1.0f);
  float4 cameraPos = mul(imagePos, instrinsicsInverse);

  // project from camera to world space using the extrinsic matrix
  float4 worldPos = mul(cameraPos, extrinsicsInverse);
  ...
}
```

The point cloud in world space needs to be transformed to the camera space of the virtual, head-tracked camera (described in detail in the next Section 3.3.3). From the point in view space, vertices will be extracted by a fixed scale value and expanded further by the sampled depth value. However, before expanding the point, the corresponding color for each point can be sampled from the color texture and added to the output structure that will be processed by the pixel shader:

[12]In this case the variables u and v range from 0 to the width (640) and height (480) of the texture and not from 0.0 to 1.0 as it is common for texture coordinates.

```
struct PixelShaderInput { float4 Pos:SV_POSITION; float4 Col:COLOR; };

Texture2D<float4> colorTex  : register(t1);
SamplerState      colorSampler : register(s0);

cbuffer constantBuffer : register(b0)
{
  ...
  matrix viewMatrix;  // virtual camera view matrix
  matrix projectionMatrix;  // virtual camera projection matrix
};
...
// expand quad to create continuous surface
static const float quadSize = 2.5f;
static const float4 quadScale = float4(1.0/depthWidth, 1.0/depthHeight,
    0.0, 0.0) * quadSize;

void GeometryShader(point GeoShaderInput vertex[1], uint primID :
    SV_PrimitiveID, inout TriangleStream<PixelShaderInput> triStream)
{
  // transform the acquired point to its position in world space
  ...
  // transform from world space to space of virtual, head tracked camera
  float4 viewPos = mul(worldPos, viewMatrix);

  // expand quad based on depth - points farther away will scale up
  float4 viewspaceScale = quadScale * depth;

  // output variable holding the quad
  PixelShaderInput output;

  // base color texture sample lookup coords, in [0,1]
  float2 depthWidthHeight = float2(depthWidth, depthHeight);
  float2 colorTexCoords = texCoord.xy/depthWidthHeight;
  // sample the color texture
  output.Col = colorTex.SampleLevel(colorSampler, colorTexCoords, 0);

  // extract a fixed-size quad
  ExtractQuad(viewPos, viewspaceScale, triStream, output);
}
```

Each point is visualized by extracting four vertices in viewspace by the determined scale value to form a fixed-size quad. The resulting vertices are multiplied with the projection matrix of the head-tracked camera (cf. Section 3.3.3) and added to the output `TriangleStream` for rendering:

```
static float4 quadOffsets[4] =
{
    float4(-0.5f, -0.5f, 0.0f, 0.0f),
    float4( 0.5f, -0.5f, 0.0f, 0.0f),
    float4(-0.5f,  0.5f, 0.0f, 0.0f),
    float4( 0.5f,  0.5f, 0.0f, 0.0f)
};

```

```
void ExtractQuad(float4 viewPos, float4 viewspaceScale, inout
    TriangleStream<PixelShaderInput> triStream, PixelShaderInput output)
{
  // expand the current point into four vertices
  [unroll]
  for (uint i = 0; i < 4; ++i)
  {
    // expand quad in view space - it always faces the camera
    float4 viewPosExpanded = viewPos + quadOffsets[i] * viewspaceScale;
    // project vertex onto virtual camera
    output.Pos = mul(viewPosExpanded, projectionMatrix);
    // append vertex to output triangle stream
    triStream.Append(output);
  }
}
```

The resulting quads are colorized in the pixel shader with the RGB values sampled from the color texture:

```
struct PixelShaderInput { float4 Pos:SV_POSITION; float4 Col:COLOR; };

float4 PixelShader(PixelShaderInput input) : SV_Target
{
    return float4(input.Col.rgb, 1.0);
}
```

Since each quad is expanded in view space, it always faces the camera[13]. In addition, because quads are scaled based on their corresponding depth they increase in size with increasing distance to the camera. Therefore, the reconstructed point cloud appears as a continuous surface although it consists of numerous planes. Due to the efficient point-based visualization the two point clouds, coming from each Kinect, can simply be rendered on top of each other without the need for any merging techniques (cf. Figure 3.14).

Render to Texture

The final point cloud is not going to be rendered to the screen right away but will undergo further post-processing (cf. Section 3.3.4). Since all the post-processing will be done on the CPU the output will have to be copied from the GPU to a texture that is accessible by the CPU. In *DirectX* the buffer for rendering to a texture instead of the back buffer can be set via a `Render Target`[14] [44, Ch. 4.1.6]. Making the GPU rendering modifiable for post-processing comprises of the following steps:

1. At first, a texture needs to be created with the `BindFlag` set to `D3D11_BIND_RENDER_TARGET`. This will allow the texture to be set as a `Render Target`.

[13]Similar to a billboard: http://en.wikipedia.org/wiki/Sprite_(computer_graphics)
[14]The off-screen rendering equivalent in *OpenGL* would be a *FrameBuffer Object*

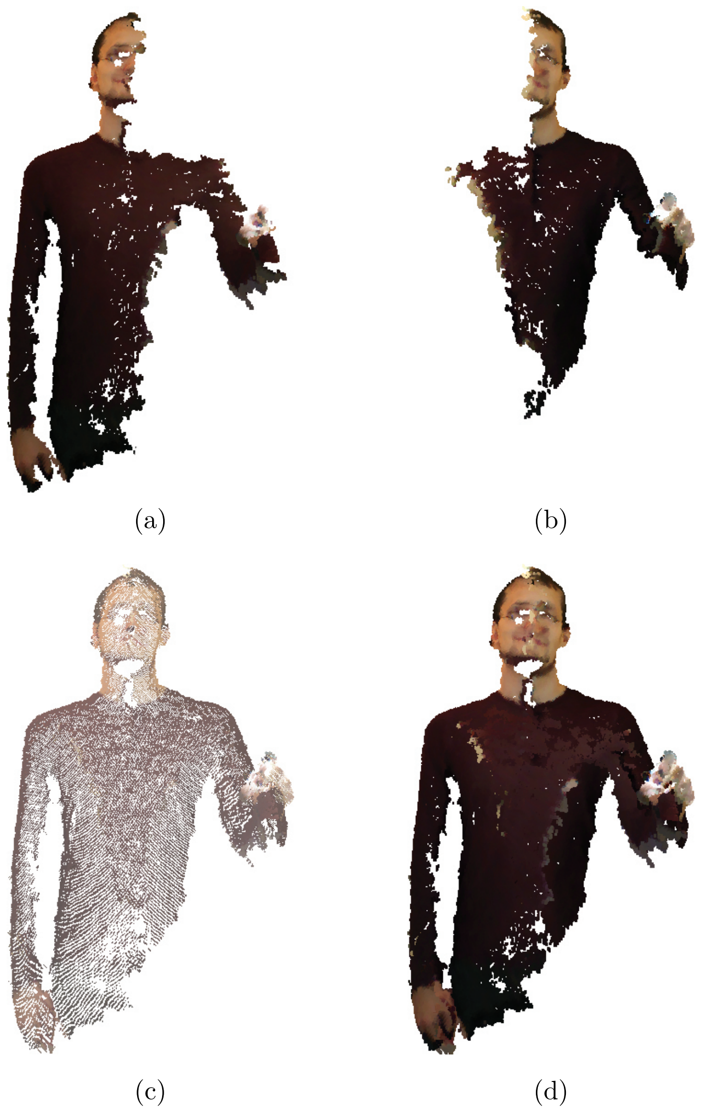


Figure 3.14: The transformed output of the left (a) and right (b) Kinect are merged and rendered as fixed size quads. For small sized quads (c) the embodiment shows holes, while it appears as a continuous surface with bigger quads (d).

2. Thus, the point clouds can be rendered to the texture by setting it as the corresponding `Render Target`.
3. Since this texture is stored on the GPU's memory it cannot be accessed by the CPU. Therefore, a second texture needs to be created with the `Usage` flag `D3D11_USAGE_STAGING` and the `CPUAccessFlags` set to `D3D11_CPU_ACCESS_READ` to support data transfer from GPU to CPU.

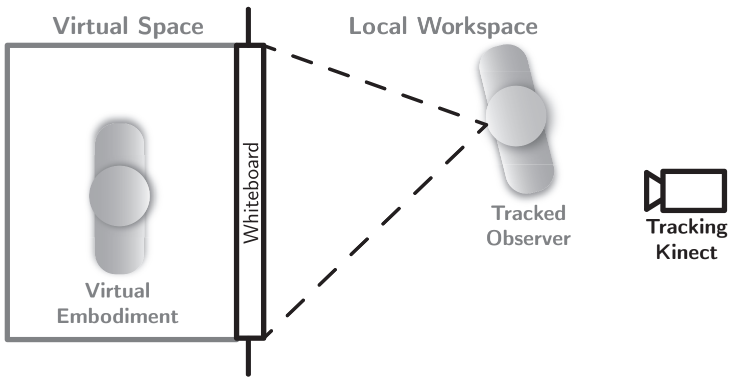


Figure 3.15: Tracking the observer's head provides motion parallax effects.

4. The content of the `Render Target` can be copied to the `STAGING` texture by calling `DeviceContext::CopyResource()`.
5. Finally, issuing `DeviceContext::Map()` maps the `STAGING` texture to RAM, providing access to the pixels for post-processing on the CPU.

3.3.3 Head Tracking

As already mentioned when discussing the reconstruction of the scene in Section 3.3.2, the view matrix, used to position the virtual camera, and the perspective matrix, used for projecting the 3D scene onto the 2D canvas, are determined by tracking the observer's head.

Positioning the Virtual Camera

By using the Kinect SDK's skeletal tracking function, the head position is acquired from the third sensor (cf. Section 3.1.2) placed behind the user (cf. Figure 3.15). `XMMatrixLookAtLH(EyePos, FocusPos, UpDir)`[15] is used to build the view matrix for the left handed coordinate system out of the (x, y, z) coordinates of the detected head joint:

```
void UpdateViewMatrix(float headX, float headY, float headZ)
{
  XMVECTOR Eye = XMVectorSet(headX, headY, headZ, 0.0f);
  XMVECTOR At = XMVectorSet(headX, headY, 0.0f, 0.0f);
  XMVECTOR Up = XMVectorSet(0.0f, 1.0f, 0.0f, 0.0f);
  View = XMMatrixLookAtLH(Eye, At, Up);
}
```

Since the `Eye`-vector and the `At`-vector share the same x and y coordinate and z of `At` is set to 0, the view vector of the camera always faces perpendicular to the screen.

[15] *OpenGL's* equivalent would be `gluLookAt`.

Off-axis perspective projection

Because the user is free to move in front of the screen, the viewing position is not necessarily at the center of the whiteboard. Therefore, the commonly used perspective projection given by `XMMatrixPerspectiveFovLH( FovAngleY, AspectRatio, NearZ, FarZ)`[16] cannot be utilized to project the 3D scene onto the screen, since it assumes that the camera is positioned along the screen axis s_n. This axis is perpendicular to the screen and centered at the screen space origin (cf. Figure 3.16). Thus, the viewing frustum must be calculated for the needed off-axis projection based on the observer's head position [67]. With the tracking Kinect placed precisely at the center of the whiteboard, the head's position $h = \begin{pmatrix} h_x, & h_y, & h_z \end{pmatrix}$ is equal to the apex of the viewing frustum (cf. Figure 3.16). The viewing frustum's left l, right r, top t and bottom b extents can thus be calculated by using the screen's width s_w and height s_h. As the viewing frustum is specified from the predefined near n to the far plane f the boundaries have to be scaled back from the screen's distance h_z to the near plane by using similar triangles:

$$l = \left(\frac{s_w}{2} + h_x\right) \cdot \frac{n}{h_z}, \qquad r = \left(\frac{s_w}{2} - h_x\right) \cdot \frac{n}{h_z}, \tag{3.23}$$

$$t = \left(\frac{s_h}{2} + h_y\right) \cdot \frac{n}{h_z}, \qquad b = \left(\frac{s_h}{2} - h_y\right) \cdot \frac{n}{h_z}. \tag{3.24}$$

Based on the frustum extents, the projection matrix

$$P = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix} \tag{3.25}$$

can be built using the function `XMMatrixPerspectiveOffCenterLH(Left, Right, Bottom, Top, NearZ, FarZ)`[17]. The viewing frustum lies in the xy-plane. The virtual camera is perpendicular to the screen and is equal to the apex of the frustum.

Since only the head joint was used for tracking more accurate methods exist [47]. However, the basic motion parallax effect is convincing (cf. Figure 3.17).

3.3.4 Post-Processing

The rendering of the virtual embodiment is of inferior visual quality (cf. Figure 3.18(a)). The virtual embodiment shows holes where depth readings are missing and at the seam of both point clouds. The latter is caused by the

[16] *OpenGL's Utility Library* equivalent would be `gluPerspective`.

[17] The *OpenGL* equivalent is `glFrustum`

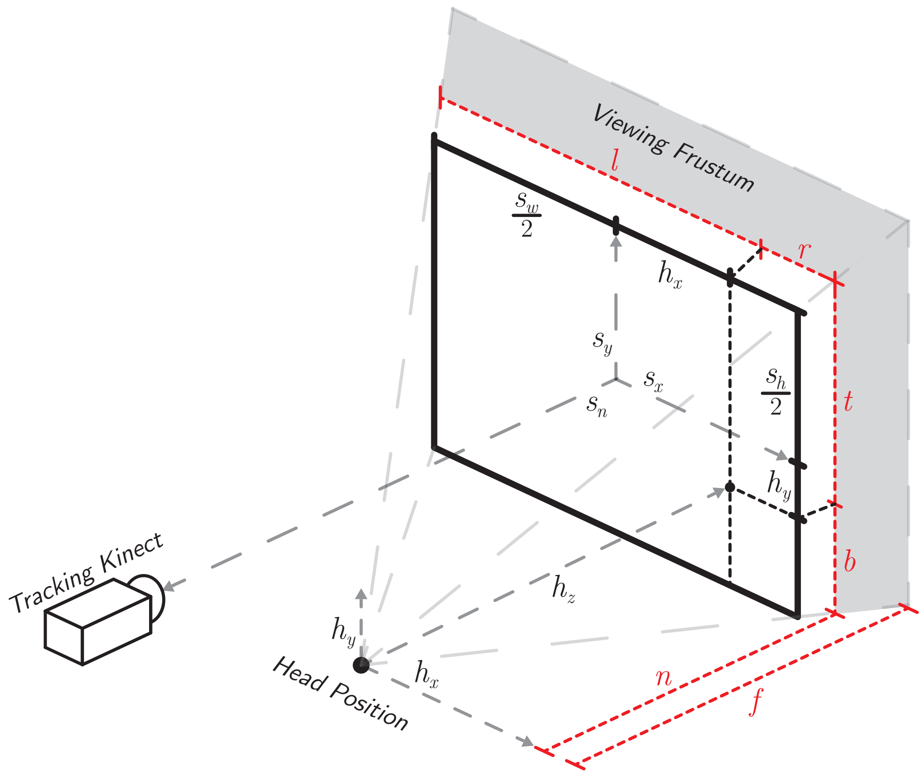


Figure 3.16: The viewing frustum of the off-axis perspective projection. Adapted from [67].

interference of the two Kinect sensors (cf. Section 1.2) although the overlap of both dot patterns is rather small.

The post-processing stage tries to beautify the results of the first render pass. Since the output was rendered to a staging texture (cf. Section 3.3.2) that can be accessed by the CPU, the image processing techniques to fill holes were implemented using C++ and *OpenCV 2.4.8*. The hole filling algorithm tries to detect small gaps in the rendering that were not removed by the median filter (cf. Section 3.3.1). Those gaps are subsequently filled using an inpainting[18] technique.

Connected Component Analysis

In order to find holes in the virtual embodiment, a connected component analysis[19] is performed. It was implemented using the *cvBlob 0.10.4*[20] library that builds on *OpenCV* and uses the algorithm of Chang et al. [11]. Af-

[18]http://en.wikipedia.org/wiki/Inpainting
[19]http://en.wikipedia.org/wiki/Connected-component_labeling
[20]https://code.google.com/p/cvblob/

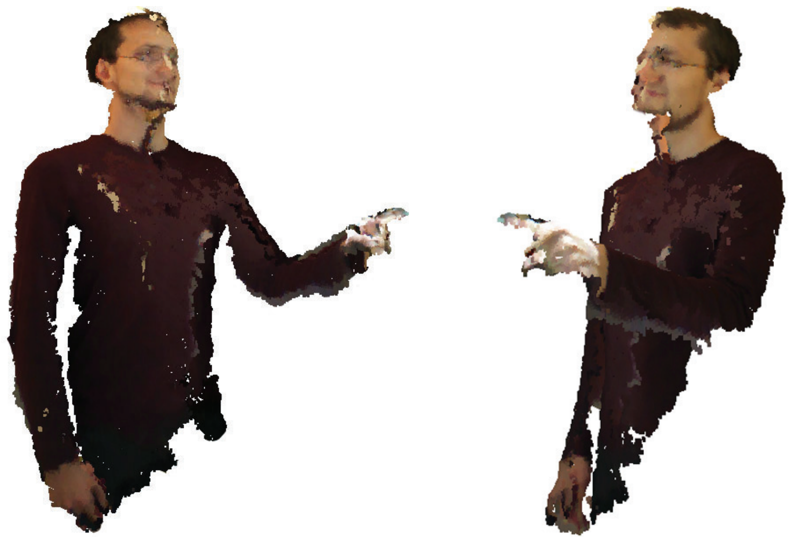


Figure 3.17: The same scene rendered from the left (a) and from the right (b) as it can be perceived by an observer moving in front of the whiteboard.

ter transforming the rendered output to a binary image (cf. Figure 3.18(b)), the connected component analysis scans for blobs of connected pixels. Only blobs smaller than a threshold λ of connected pixels are detected and labeled (cf. Figure 3.18(c)). To avoid an unwanted merging of disconnected regions, λ is empirically chosen as a low value of 1500 pixels for a screen resolution of 1280×800 pixels. The detected blobs are equal to the small holes in the rendered virtual embodiment, since the algorithm is scanning for the background pixels. The contours of the holes can be fuzzy and of ambiguous colors. Thus, a morphologically dilation is applied onto the binary image to enlarge the selected holes (cf. Figure 3.18(d)) and perform the filling with more reliable contour pixels.

Hole Filling

Filling the detected holes is performed by using *OpenCV's* `inpaint` method that is based on Telea [58]. The binary image of the blobs is utilized as an inpainting mask. The algorithm gradually fills up the boundaries of the detected blobs in the source image. It inpaints each pixel by a weighted sum of all neighboring pixels.

For small holes enough boundary information is given and the reconstruction of the embodiment achieves good results (cf. Figure 3.18(e)). However, if a too large threshold is used, a merging of separated regions can occur (cf. Figure 3.18(f)). The complete sequence is given in Listing 3.2.

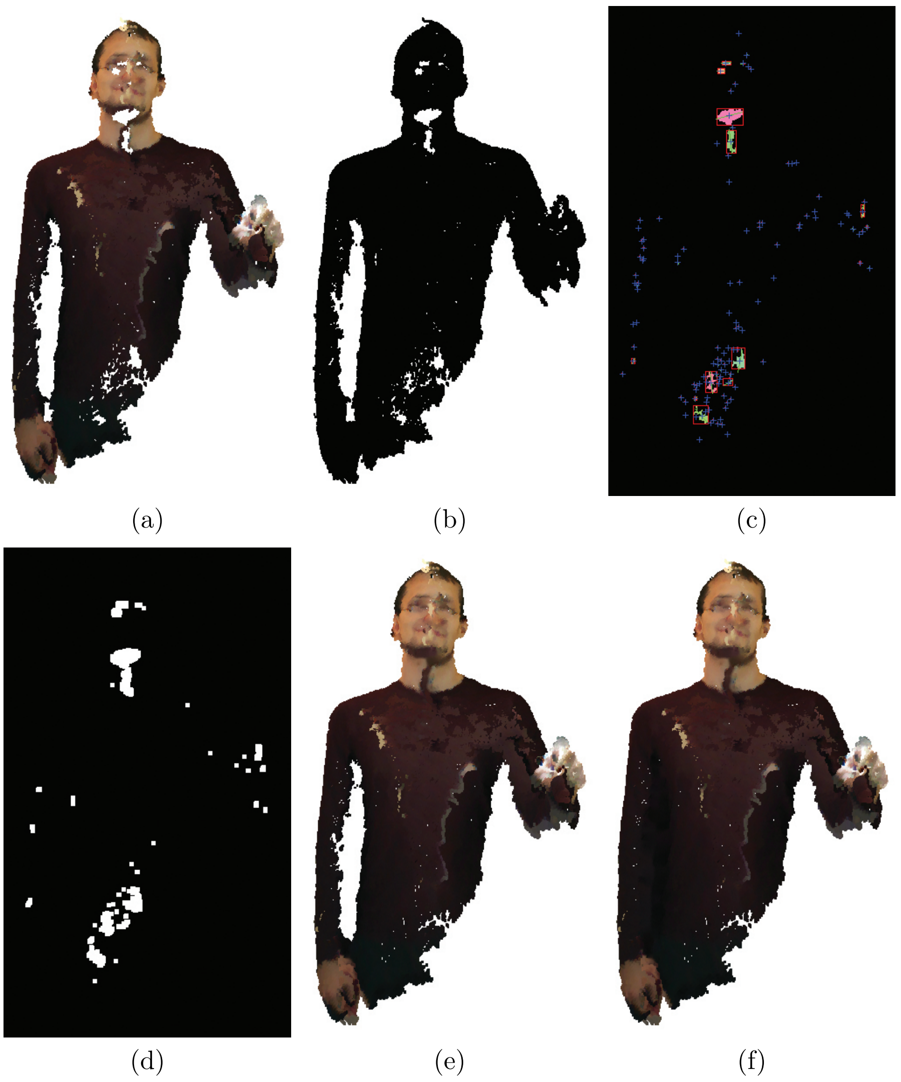


Figure 3.18: The sequence of filling holes in the post-processing stage. The original rendering output (a) is transformed to a binary image (b) to perform a connected component analysis (c). The binary image with the detected blobs is morphological dilated (d) to use them as an inpainting mask. For small holes the inpainting achieves good results (e) while for larger thresholds also the space in between the arm and the body is filled (f).

3.3.5 Merging Embodiment and Application

Superimposing the embodiment on top of any shared content to raise the social presence is a primary goal of *3D-Board*. Thereby, the virtual embodiment should be rendered independently of any other application. To achieve this goal, the output is displayed in a WPF window, written in C#. This

Algorithm 3.2: Hole filling algorithm.

```
    Detect and fill holes in the RGB image $I$ using the threshold $\lambda$.
 1: HoleFilling($I, \lambda$)
 2:     $I_g \leftarrow$ ConvertGrayscale($I$)          ▷ Grayscale for binary threshold
 3:     $I_{bw} \leftarrow$ Threshold($I_g, 0$) ▷ BG is black, binary threshold values $> 0$
 4:
 5:     $B \leftarrow$ DetectBlobs($I_{bw}$)                          ▷ Get list of blobs
 6:     $I_b \leftarrow 0$                                      ▷ Create binary blob mask
 7:     for all $b \in B$ do
 8:         if $b_{size} \leq \lambda$ then
 9:             AddBlob($b, I_b$)          ▷ Add blobs smaller than $\lambda$ pixels to $I_b$
10:         end if
11:     end for
12:     $I_b \leftarrow$ MorphDilate($I_b$)
13:     $I \leftarrow$ Inpaint($I, I_b$)        ▷ Inpaint image $I$ using the binary mask $I_b$
14: end
```

window is set to being transparent, borderless and the topmost element on the screen. In addition, it does not capture input events of any kind (mouse, keyboard or digital pen) but passes the events to the underlying application. Thus, the virtual embodiment can be superimposed on top of any content and is independent of the shared workspace.

Application Integration

The native rendering application is compiled as a *Dynamic Link Library*[21] and is invoked by the managed client application by using the *Platform Invocation Services*[22]. Interoperability between WPF and DirectX is accomplished via the `D3DImage` element. Embedded into the output window, this element functions as an alternative input source for a WPF `Image` element. It can host a `Direct3D Surface` (equal to a render buffer) and triggers an update of the `Image's` content after each render cycle (cf. Figure 3.19).

Thus, the post-processed texture can simply be copied to a render target that is then used as the input for the `D3DImage` element. The `D3DImage` does support transparency to blend the virtual embodiment with the underlying content by rendering the remote user semitransparent. Additional effects, such as adding an opacity-gradient to the superimposed embodiment, can easily be applied in WPF (cf. Figure 3.20).

[21]http://de.wikipedia.org/wiki/Dynamic_Link_Library
[22]http://msdn.microsoft.com/en-us/library/aa288468(v=vs.71).aspx

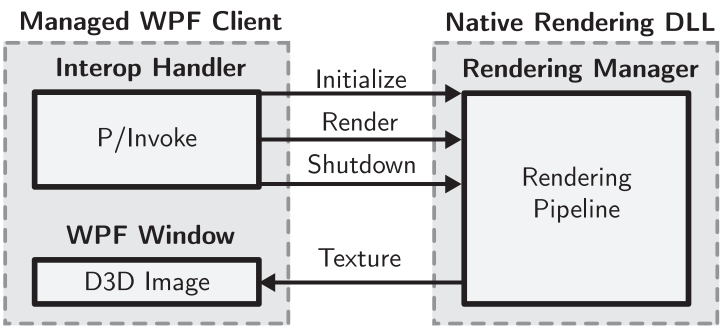


Figure 3.19: The interoperability between the native rendering application and the managed client.

3.3.6 Results and Performance

The presented rendering pipeline transforms the color and depth data of two Kinect sensors into a single virtual representation of a remote user that supports gaze awareness, facial expressions and gestures. This concluding section recapitulates the individual rendering steps and depicts their performance impact.

Step by Step

Considering the simplicity of the frame differencing approach for removing the background (cf. Figure 3.21 (a)) it is very effective. The remaining noise is mostly present in the distant background that is not perceivable by the observer.

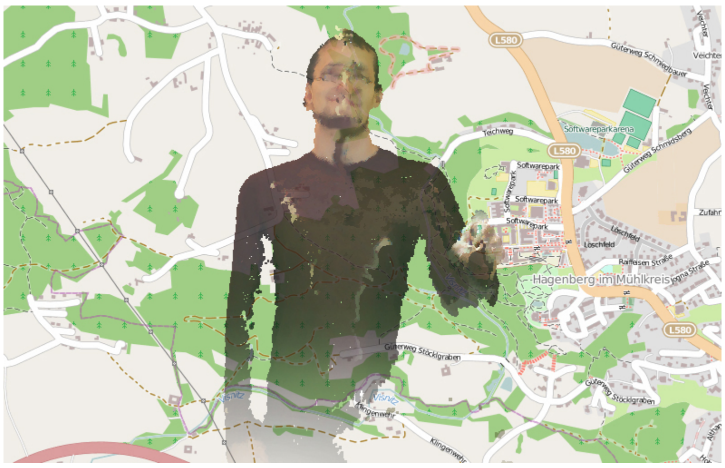


Figure 3.20: Additional effects, such as 75% transparency and an opacity-gradient at the bottom are applied for superimposing the embodiment.

Table 3.1: The frames per second for the raw point cloud, the background removal, the median filtering, WPF transparency effects and the inpainting (average and large hole sizes) for different display resolutions.

Display Resolution	Raw	Remove BG	Median Filter	WPF Effects	Inpaint Avg.	Inpaint Large
1024 × 768	67	70	58	58	25	22
1280 × 800	67	69	56	55	22	20

Although utilizing the Kinect SDK to match the colors (cf. Figure 3.21 (b)) of the individual cameras can achieve good results, it is rather unreliable. The sensors are very sensitive to changing light conditions and a more sophisticated implementation would be needed, as discussed in Section 3.3.1.

The raw point cloud (cf. Figure 3.21 (c)) does show holes for missing depth readings. Small holes often occur due to the noisy depth maps. Larger gaps are the result of blind spots for example at the throat, the eyeholes or where the arms occlude the body. Also the interference between the structured dot patterns at the seam of both point clouds can generate incomplete depth maps.

Applying a median filter (cf. Figure 3.21 (d)) to the depth maps is effective in filling very small holes and smoothing the depth readings without altering the appearance of the embodiment. In addition, this leads to a reduction of background noise since also the reference depth map is filtered. However, larger gaps can only be filled by the inpainting step (cf. Figure 3.21 (e)). Since the inpainting is applied to the final output, as it is seen by an observer, it is very successful in filling all visible holes as long as they are small in size and enclosed by valid data. In rare cases inpainting leads to inferior visual quality if the surrounding color quality is poor or unwanted mergings occur. A better option would be to add more cameras to gain additional depth and color readings. However, this comes at other costs such as a more complicated setup and calibration process as well as additional performance penalties.

The Price of Beauty

The extensive rendering pipeline comes at a high performance cost. Table 3.1 shows the frames per second for each rendering step added to the pipeline executed on the hardware described in Section 3.1.1.

Rendering the point cloud as billboards in the geometry shader is rather efficient. The simple background removal provides a performance gain, since the depth readings in the background are discarded and thus no quads need to be created and rendered. The 3×3 median filter is very costly since it

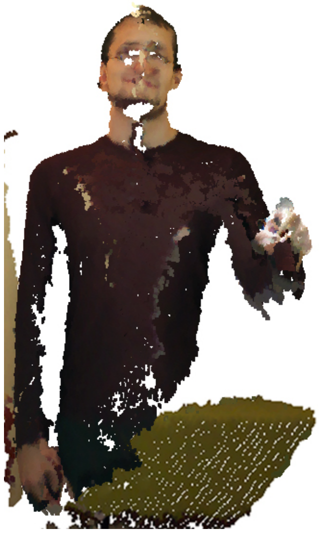
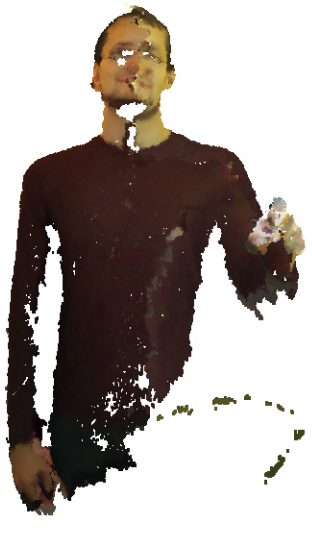
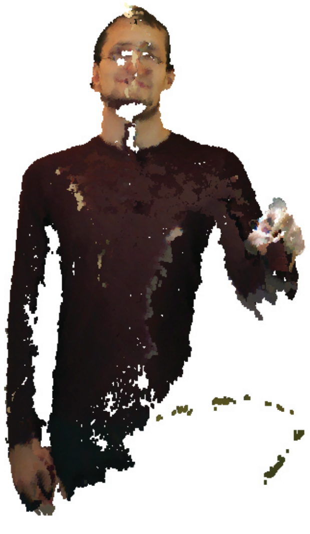
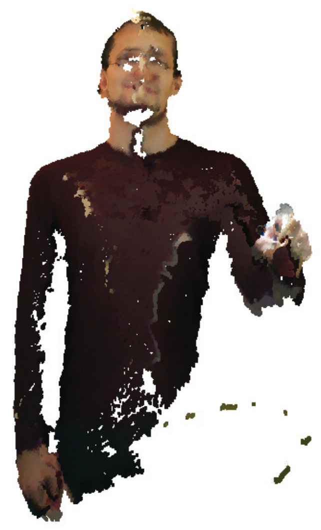
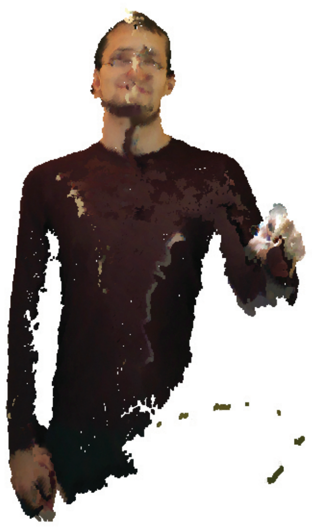
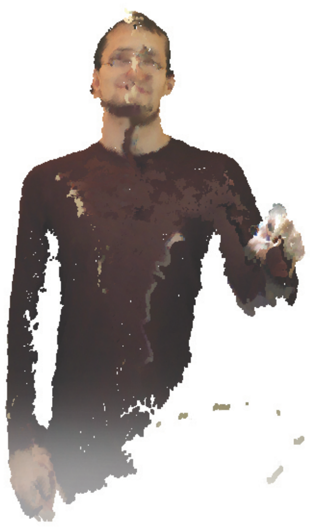

(a)

(b)

(c)

(d)

(e)

(f)

Figure 3.21: Removing the background (a) and matching the color (b) is essential for rendering the calibrated point clouds (c). In addition, the median filter fills small holes, smooths the depth map and thus also removes some noise (d). Larger gaps are inpainted (e) and the transparency effects are added (f) for merging the embodiment with an application.

is executed on the CPU for each depth map. So far, each step depended solely on the Kinect's depth resolution. The screen's resolution had very little influence. The transparency effects added to the WPF output and the inpainting are applied to the full size of the whiteboard. Therefore the performance penalty increases with the output resolution. While the cost of the WPF effects is almost negligible, most frames are lost to the hole filling approach. It strongly depends on the number of pixels on the screen and the hole count. Thus, it was tested for two different quantities of holes. The average hole count refers to the filling as seen in Figure 3.18(e), while the larger count is equal to Figure 3.18(f). The main cause of the large performance impact is the CPU based implementation. The higher the resolution and hole count, the more pixels the single execution thread has to fill.

The immersive effect of the virtual embodiment is already convincing without the hole filling at a very high performance. However, in order to achieve full fidelity and a stable frame rate with an increased display resolution, parallelization or a GPU based implementation of the median filter as well as the inpainting is inevitable.

Chapter 4

Demo Applications

Two applications were developed to demonstrate the capabilities of *3D-Board*. The two programs differ in the way the remote user's embodiment was integrated (cf. Figure 4.1). Both scenarios were built on existing software, written in C# and WPF. Because the applications were tested using the setup described in Section 3.1.1, the shared data was simply mirrored on both whiteboards while the virtual embodiment was superimposed on top of only one screen.

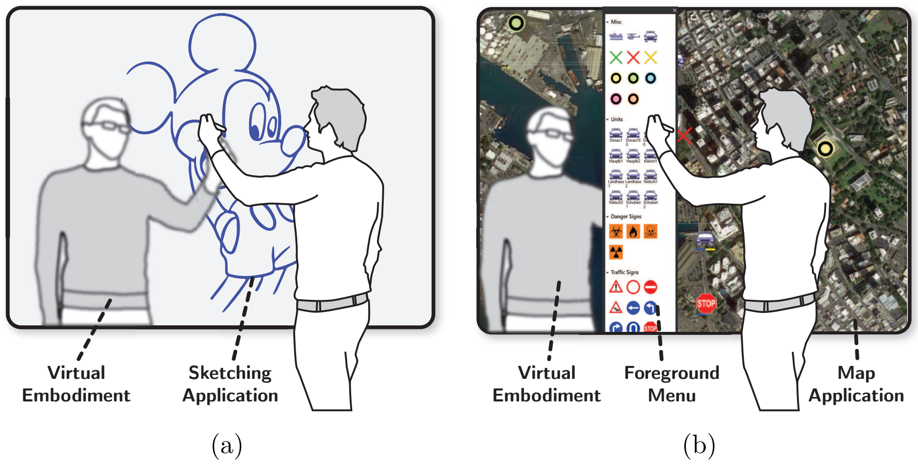


Figure 4.1: While in the first demonstration the virtual embodiment is superimposed on top of the shared data, in the second application the visualization is embedded in between the content and important menus.

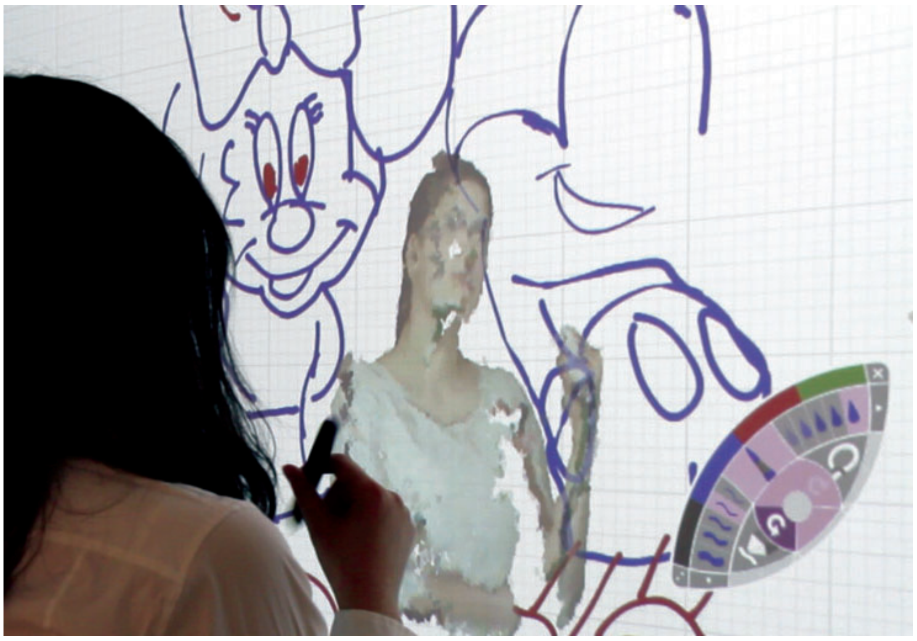


Figure 4.2: *3D-Board* enables shared drawing activities and ideation.

4.1 Remote Sketching

The first application consists of a simple remote sketching tool (cf. Figure 4.2). The users can utilize the shared drawing area for creating new ideas. In order to improve the coordination, the transparent virtual embodiment is superimposed on top of the entire content, as described in Section 3.3.5. Due to the raised awareness, users can manage their workspace as they would to in a co-located setup.

4.2 Map Surveillance

The idea of *3D-Board* has its background in a research project conducted in close collaboration with the local police forces. For large scale operations, rescue teams need to trace their units in the field and keep an overview of the situation. Therefore, the available data needs to be shared between all geographically distributed teams. An intuitive Common Operational Picture (COP) was developed that is based on an interactive map application for large scale, digital whiteboards. With simple gestures relevant data such as labels, text, signs and pictures can be added the shared map at their exact location. Thus, the distributed users have to collaborate to compile the COP.

The remote embodiment is again superimposed on top of the interactive map. However, in order to enable an effective cooperation between all par-

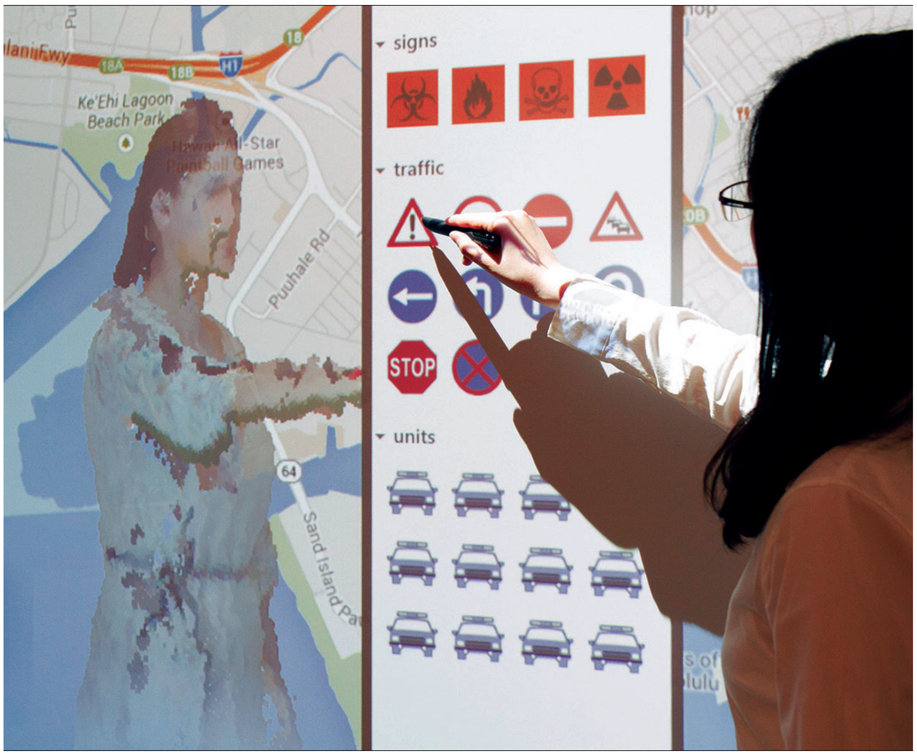


Figure 4.3: *3D-Board* can be used to collaboratively keep track of crucial operational information. To increase the social presence of the remote user, the virtual embodiment is superimposed on top of the shared content, while important elements are not occluded.

ties, important menus for selecting tools and items should never be occluded. Therefore, *3D-Board* is directly integrated into the application. This allows crucial elements to stay visible, while the virtual embodiment is embedded in between the COP (cf. Figure 4.3) and the menus.

3D-Board is easy to use and easy to integrate into existing applications. In addition, it is also very effective and improves the remote collaboration, as the evaluation in the next chapter proves.

Chapter 5

Evaluation

This chapter presents the empirical user study conducted to explore the benefits and limitations of *3D-Board*. In two experiments the performance and effectiveness were put to a test and compared against different techniques. This chapter gives an in depth explanation of the study design and states the results of each experiment. In addition, the last section will reason about the conclusions drawn from observations and the interviews.

5.1 Study Design

All participants conducted both experiments using the same apparatus and techniques. Each participant was given an overview of the project and the experiment procedure. After filling out a background questionnaire, they were given time to practice with the techniques until they felt comfortable with the system. Participants were asked to complete each trial as fast and as accurately as possible. For both experiments computer logs captured the pen input and selections for evaluation. Preference data were collected through post-experiment and exit questionnaires. Both experiments were implemented using C# and WPF

5.1.1 Participants

In total, 12 paid students (6 female, 6 male) participated. Their age ranged from 20 to 29 years (Mean(M) = 24.08, Standard Deviation(SD) = 2.53). Participants performed the study in pairs of two. One was acting as an *instructor* giving commands to the *operator*. The *operator* on the other hand, had to interpret the given instructions and act accordingly. After finishing all trial runs for one constellation, participants switched roles in order to act as both, the instructor and the operator, for all experiments.

An important aspect of the study was to find out whether users felt comfortable with a technique when cooperating very closely with their partner.

For the course of the study it was therefore crucial that the group members had never met each other before. This increased the impact of close collaboration on the proxemic behavior, as defined by Hall [25, p. 113–129]. Proxemics describes how the physical distance between people can influence the interpersonal communication. Two users, not knowing each other, are thus more likely to show defensive behavior when they feel their personal space is being violated.

While 75% of the participants reported not being familiar with interactive whiteboards, 6 out of 12 of the participants had more experience with pen-based interfaces. Finally, 67% of the participants are using voice-over-IP application on a weekly or even daily basis.

5.1.2 Apparatus

The study was conducted in a quiet room using an extended apparatus (cf. Figure 5.1) of the setup described in Section 3.1. The workplaces were separated from each other by a metal frame with blinds for visual cover. Thus, participants were unable to see the other whiteboard, but could communicate via voice commands and the technique in use. In addition, a fourth Kinect camera was placed behind the remote instructor. This sensor was solely used during evaluation for the *2D-Back-Facing* technique and was not part of the *3D-Board* setup.

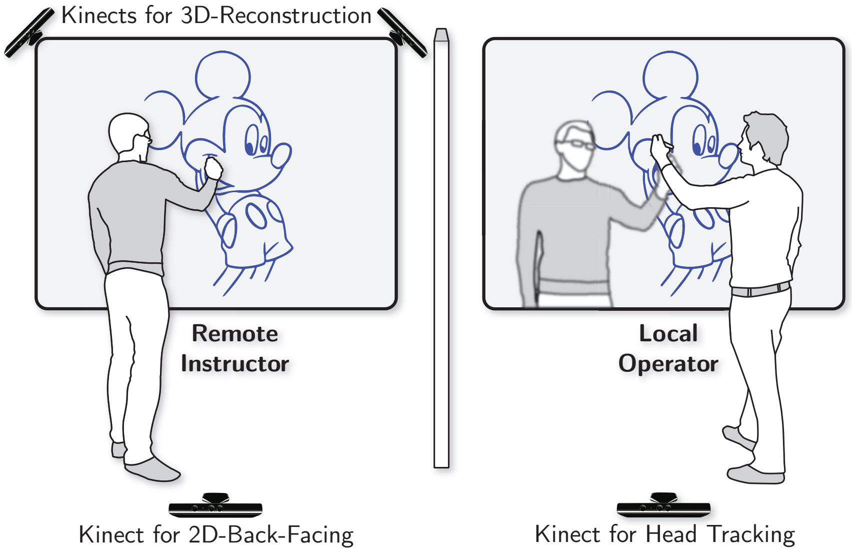
5.1.3 Techniques

Both experiments were conducted using the techniques *3D-Board*, *2D-Back-Facing* and *Co-Located*.

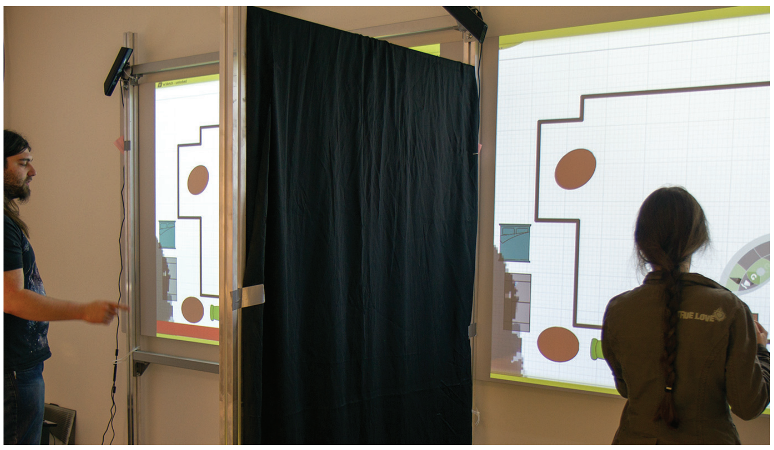
Technique 1: *3D-Board*

When using the *3D-Board* technique a front-facing 3D embodiment of the instructor could be seen on the operator's whiteboard (cf. Figure 5.2). Additionally the perspective view of the operator changed according to the head-tracking of the operator. Thus the operator and the instructor were facing each other through the whiteboard and could simultaneously work on the same spot on the whiteboard without any interference. It remained to observe, however, if the operator would feel a violation of the personal space when working in close proximity.

The *3D-Board* technique used during the evaluation is based on an earlier implementation presented in [65]. While the usage and interaction technique are the same for the earlier implementation as well as the approach presented in Chapter 3, the latest proof-of-concept implementation does not meet the performance requirements to guarantee a stable frame rate (cf. Section 3.3.6). Thus, to conduct an encumbrance free evaluation the earlier implementation was chosen for conducting the study since it ensures a 30Hz



(a)



(b)

Figure 5.1: The sketch (a) and the picture (b) of the apparatus as it was used for the evaluation. On the left side, the instructor is giving the orders, while on the right side, the operator has to react accordingly.

frame-rate. The visual quality of the first prototype is comparable with the latest prototype and provides enough fidelity to clearly grasp all actions of the virtual embodiment.

Technique 2: 2D-Back-Facing

In this condition a similar setup as proposed by CollaBoard [41] was used. A fourth Kinect camera captured the backside of the instructor with a video

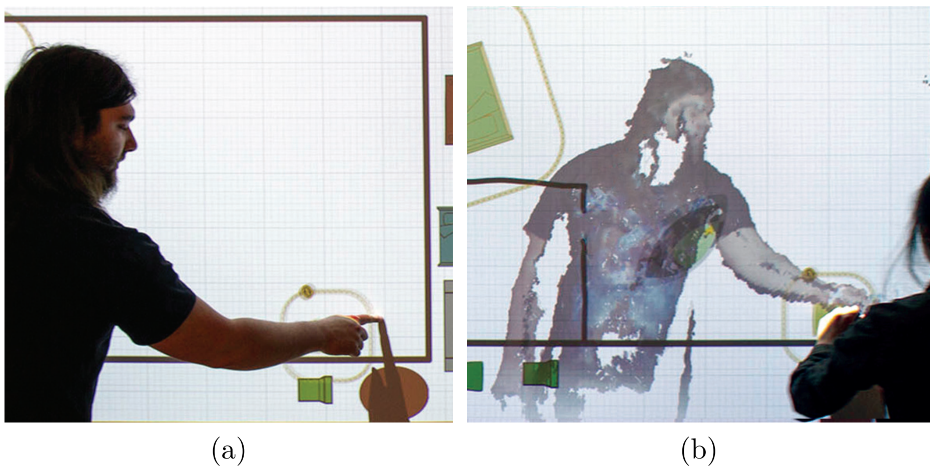


Figure 5.2: The *3D-Board* technique allowed the instructor (a) and the operator (b) to work on the same data at the same time.

resolution of 640 × 480, as depicted in Figure 5.1 (a). The instructor was separated from the background of the video image by using the Kinect for Windows SDK's[1] background removal feature. Therefore, a clean image of the instructor could be orthogonally projected onto the whiteboard. However, the orthogonal projection is often not perfectly aligned with the actual gestures of the user. Thus, in order to make the instructor aware of his own actions, the 2D embodiment was superimposed over the content of the whiteboard of both users with a transparency of 75% (cf. Figure 5.3).

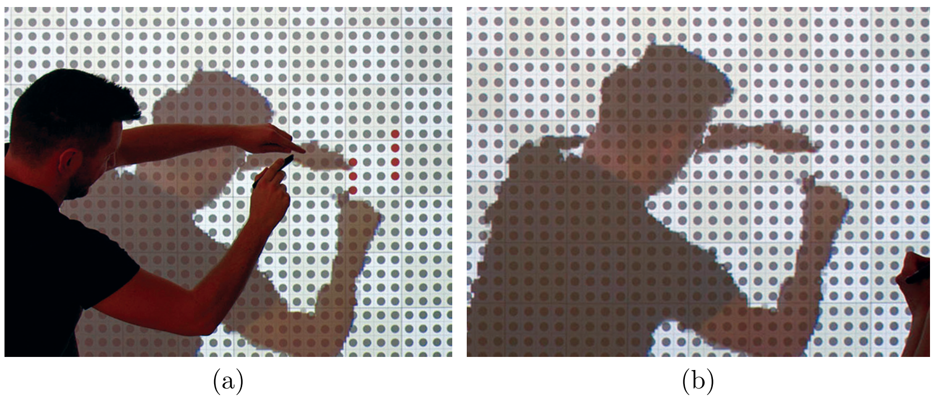


Figure 5.3: The *2D-Back-Facing* technique was visualized as the orthogonal projection of the instructor (a) onto the operator's (b) as well as the instructor's whiteboard.

[1]http://www.microsoft.com/en-us/kinectforwindows/

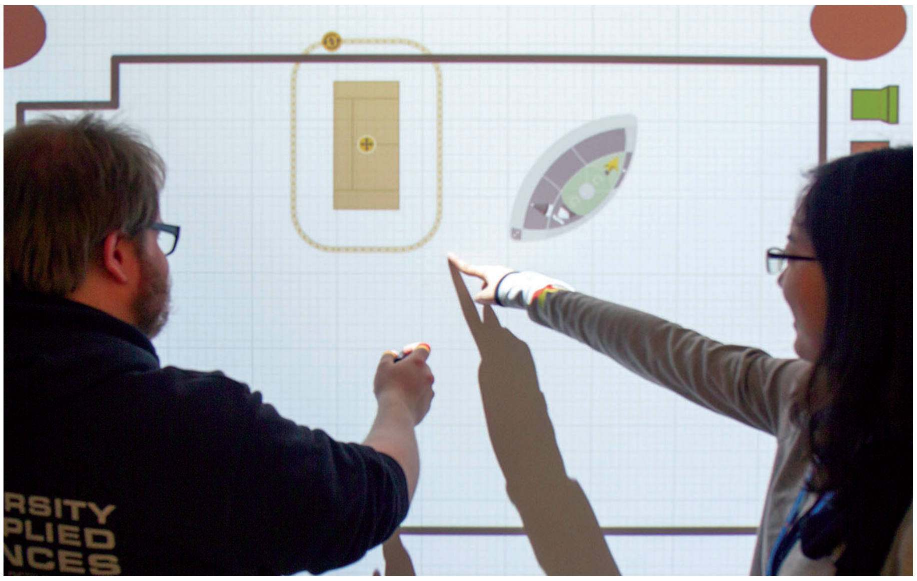


Figure 5.4: *Co-Located* was used as the ideal baseline scenario. For close collaboration users often violated each other's personal space by crossing their arms.

Technique 3: Co-Located

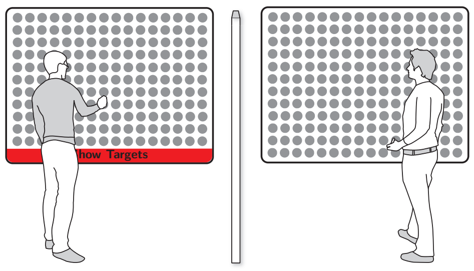
The last technique was used as a baseline for comparison. Both participants were collaboratively working in front of the same whiteboard, thus simulating a co-located situation (cf. Figure 5.4). Although this is an ideal scenario, the participants could easily invade the partner's personal space and would interfere with each other when working on the same data.

5.2 Experiment 1: Abstracted Environment

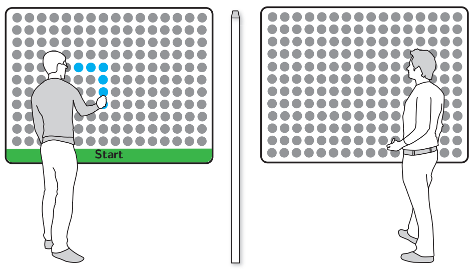
The first experiment investigated how well the participants could understand the connection between the remotely located person and the digital content. The goal of this experiment was to measure mainly quantitative data by evaluating whether the techniques differ in terms of accuracy and speed.

5.2.1 Task

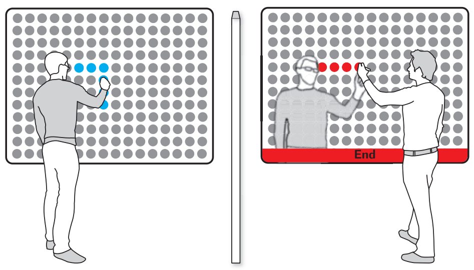
The task environment, based on the experiment of Grossman et al. [21], consists of a grid of points. A certain amount of points, arranged in different target structures, were candidates for selection. While the instructor had to show the targets to the operator by pointing at them, the operator had to select the points in question as quickly as possible. The participants were not allowed to talk during the trials in order to keep the test focused on



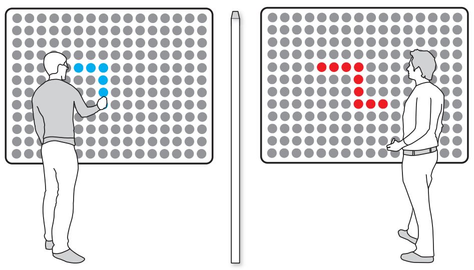
(a)



(b)



(c)



(d)

Figure 5.5: The **Show-Targets** button (a) reveals the targets for the instructor only (b). The **Start** button triggers the timer and the instructor can point out the targets (c). Once the operator is finished marking all points, the **End** button stops the timer and the number of mistakes are counted (d).

the deictic gestures. The complete procedure is illustrated in Figure 5.5 and consisted of the following steps:

1. Both participants were presented with a grid of inactive target points (cf. Figure 5.5 (a)).
2. When the instructor tapped on the **Show-Targets** button, a set of targets turned blue, only visible to the instructor (cf. Figure 5.5 (b)). When testing the *Co-Located* technique the operator turned around and faced the wall to avoid seeing the targets.
3. After memorizing the target points the instructor could press the **Start** button to activate a timer taking the time for each trial. Then the instructor was asked to point at the targets, showing the operator which targets to select (cf. Figure 5.6).
4. The operator had to select the targets as quickly as possible with a single tap on the object (cf. Figure 5.5 (c)). Once finished, the operator pressed the **End** button to stop the timer and take the time for the trial run (cf. Figure 5.5 (d)). The instructor was not allowed to correct any mistakes made by the operator.
5. The users switched roles after completing all trials for one turn.

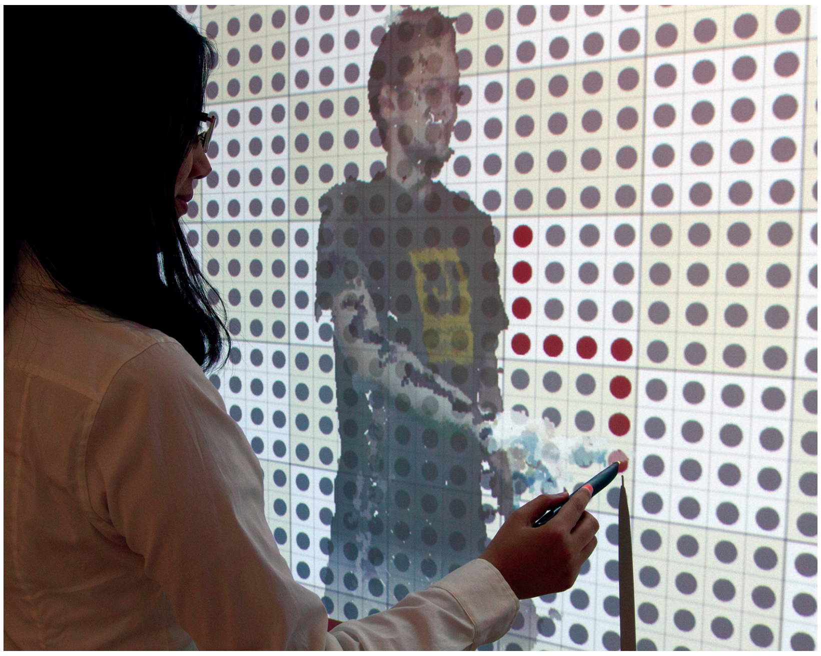


Figure 5.6: In the first experiment, the instructor pointed at target structures of different complexity levels. The operator had to mark the target points accordingly.

Two different levels of target complexities (*Simple* and *Complex*) were tested. Simple targets consisted of points arranged in a single line in either horizontal or vertical orientation. Complex objects were shaped as two or three chains that could either be linked together or separated by a small amount of inactive points (cf. Figure 5.6). The diameter of the target as displayed on the interactive whiteboard under the *Simple* condition was 3.5 cm and 2 cm under the *Complex* condition.

5.2.2 Procedure

A repeated measures within-subject design was used. Technique (*3D-Board*, *2D-Back-Facing*, *Co-Located*) and Complexity (*Simple*, *Complex*) were used as independent variables. The presentation order for the techniques was counterbalanced while the complexity level always started with *Simple* and ended with *Complex*. For each turn, a total of three trials had to be completed. After these three trials the Complexity was raised, resulting in six tests per Technique per person. Thus each group completed a total of 36 trials, resulting in a total trial count of 216:

	12	participants
×	3	Techniques (*3D-Board*, *2D-Back-Facing*, *Co-Located*)
×	2	Complexities (*Simple*, *Complex*)
×	3	task trials
	216	total trials.

A total of 18 unique target structures (3 Techniques × 2 Complexities × 3 trials = 18 targets) were repeated in a predefined order after all three techniques were tested with both complexities. Since the techniques were counterbalanced, all techniques were tested four times with each unique target. The variety and amount of target structures made it impossible for the users to remember the pattern. The procedure and sequence of trials is shown in Table 5.1.

Table 5.1: Counterbalancing of the independent variables *Technique* (CL = *Co-Located*; 2D = *2D-Back-Facing*; 3D = *3D-Board*) and *Complexity* and the 18 *Target* structures for each participant.

Part.	**Ind.Var.**	**Sequence**																	
1 & 4	Compl.	*Simple*									*Complex*								
&	Techn.	CL			2D			3D			CL			2D			3D		
7 & 10	Target	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
2 & 5	Compl.	*Simple*									*Complex*								
&	Techn.	3D			CL			2D			3D			CL			2D		
8 & 11	Target	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
3 & 6	Compl.	*Simple*									*Complex*								
&	Techn.	2D			3D			CL			2D			3D			CL		
9 & 12	Target	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

In addition, qualitative feedback was collected after each technique using user experience questionnaires. Participants had to rate the techniques based on the following criteria: easy-to-use, accuracy, and pointing awareness on a 5-point Likert scale. Task load ratings were collected using the NASA Task Load Index (TLX) questionnaires. The whole test including training sessions and questionnaires lasted for approximately 40 minutes for each group.

5.2.3 Hypotheses

The following hypotheses were defined covering overall preference, speed and accuracy of the techniques:

Hypothesis 1: 3D-Board would be at least equally fast as the Co-Located technique.
When using *3D-Board*, both users are able to work on the same area of the workspace without stepping on each other's toes. Therefore it allows for simultaneously pointing at a target and marking the target. Thus it has an advantage over the *Co-Located* setup and will perform at least equally well in terms of speed.

Hypothesis 2: 2D-Back-Facing would be the slowest technique.
The *2D-Back-Facing* technique will be the slowest technique since deictic gestures are often occluded and it will take time for the instructor to find the right posture.

Hypothesis 3: 3D-Board is more accurate than 2D-Back-Facing.
The head-tracking capability of *3D-Board* allows for a much better recognition of the other participant's gesture and posture. However, *3D-Board* will not be more accurate than the *Co-Located* technique.

5.2.4 Quantitative Results

Quantitative data, such as missed selections and trial completion times, were analyzed using a repeated measures ANOVA ($\alpha = 0.05$) separately for each level of complexity. The Greenhouse-Geisser correction was used if the assumption of sphericity was violated. Main Effects were shown by a repeated measures analyses of variance. In order to confirm or reject the formulated hypotheses post-hoc analyses on the main effects were conducted. These consisted of paired-samples t-tests with family wise error rate controlled across the test using Holm's sequential Bonferroni approach [16]. For all bar charts, the error bars indicate the range of two standard errors of the mean (above and below the mean).

Trial Completion Time

For the trial completion time, a repeated measures analysis of variance showed main effects for the *Technique* ($F_{2,10} = 20.075, p < 0.0001$) as well as for the *Complexity* ($F_{1,11} = 82.227, p < 0.0001$). Figure 5.7 depicts the overall mean time for each technique.

Post-hoc analysis showed that the *3D-Board* technique was significantly faster ($M = 13.5\,\mathrm{s}, SD = 3.3\,\mathrm{s}$) than *2D-Back-Facing* ($M = 20.98\,\mathrm{s}, SD = 6.97\,\mathrm{s}$) with $p < 0.001$ for the *Complex* task. The *3D-Board* technique is faster since natural deictic gestures are immediately understood due to the front facing view of the virtual embodiment. When using the *2D-Back-Facing* technique the body of the instructor will occlude the visualization

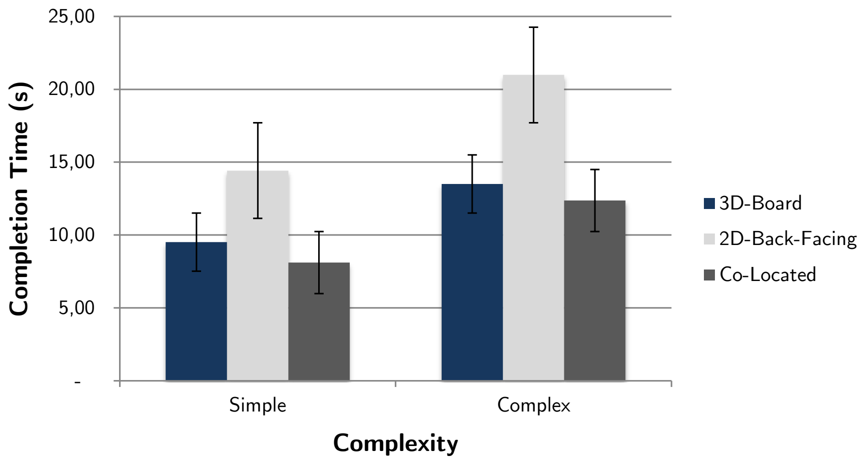


Figure 5.7: Overall completion time by Techniques and Complexity (*Simple* vs. *Complex*).

of the hands, since the camera image is projected orthogonally from the back of the room onto the whiteboard. Thus, the instructor has to find the correct posture to avoid occlusion and make the deictic gesture clear to the operator (cf. Figure 5.8). This can take a significant amount of time causing the *2D-Back-Facing* technique to be slower, thus confirming *Hypothesis 2.*

Participants performed slightly faster in the *Co-Located* scenario ($M = 12.37\,\text{s}, SD = 3.67\,\text{s}$) than in the *3D-Board* setup. A pairwise comparison, however, showed no statistical significance ($p = 0.596$). This confirms *Hypothesis 1* since *3D-Board* can be considered as fast as the *Co-Located* tech-

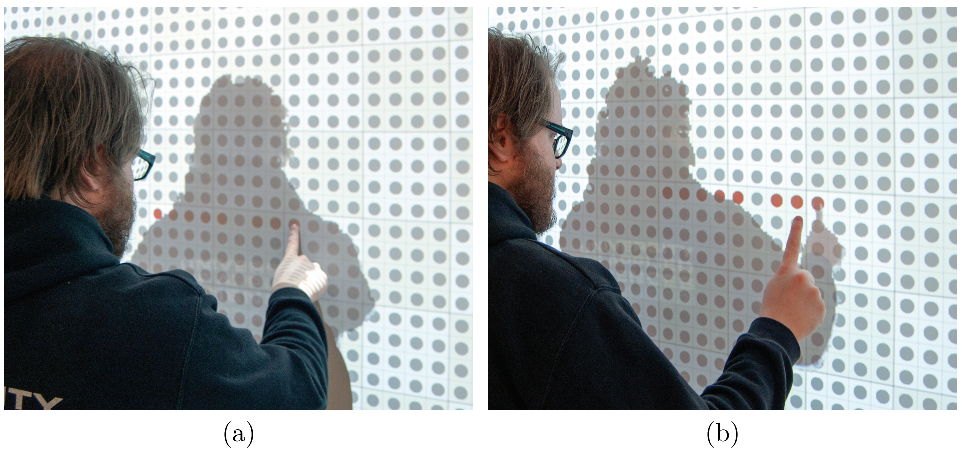

(a) (b)

Figure 5.8: In (a) the hands are occluded by the body of the participant. Only in the sideways stance the deictic gesture is clearly visible (b).

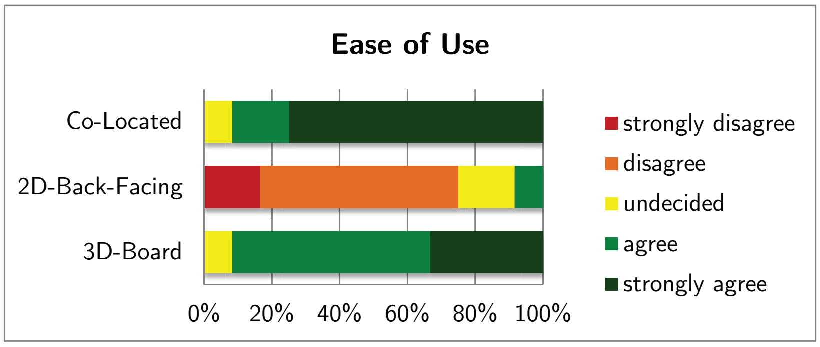
nique. Similar conclusions can be drawn for the simple task since *3D-Board* ($M = 9.52\,\mathrm{s}, SD = 4.27\,\mathrm{s}$) was significantly faster than *2D-Back-Facing* ($M = 14.42\,\mathrm{s}, SD = 4.64\,\mathrm{s}$) with $p = 0.01$ and no significant difference was measured for the *Co-Located* technique ($M = 8.22\,\mathrm{s}, SD = 2.21\,\mathrm{s}$) with $p = 0.346$.

Error rate

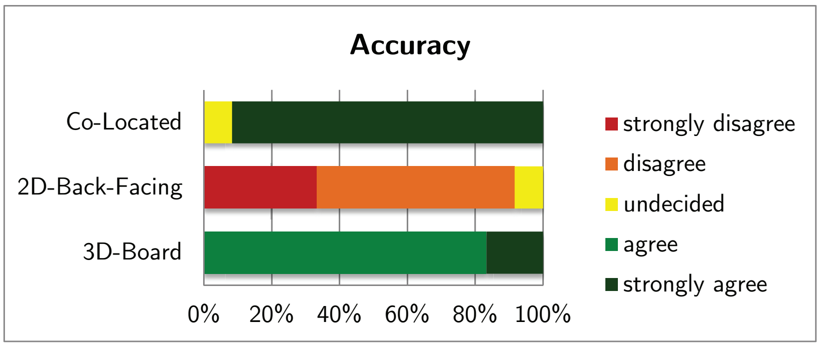
In addition to the trial times the number of errors were counted. However, out of a total of 775 target points in the *Complex* scenario only 29 targets were selected incorrectly. On average 7.44 points were candidates for selection per trial, for *Complex* structures. 7.22 ($SD = 1.08$) targets have been selected correctly using the *3D-Board* technique compared to 7.25 ($SD = 1.01$) targets using the *Co-Located* technique, and 7.06 ($SD = 1.47$) using the *2D-Back-Facing* technique respectively. Given such small error rates, no significant differences could be found for any technique with $p = 1.0$. *Simple* target structures resulted in very similar data with no significant difference. Thus *Hypothesis 3* needs to be rejected since all three techniques appear to be very accurate, given participants take their time. In addition, the visual quality of *3D-Board* can be considered as being good enough, even when interacting with very small objects.

5.2.5 Qualitative Results

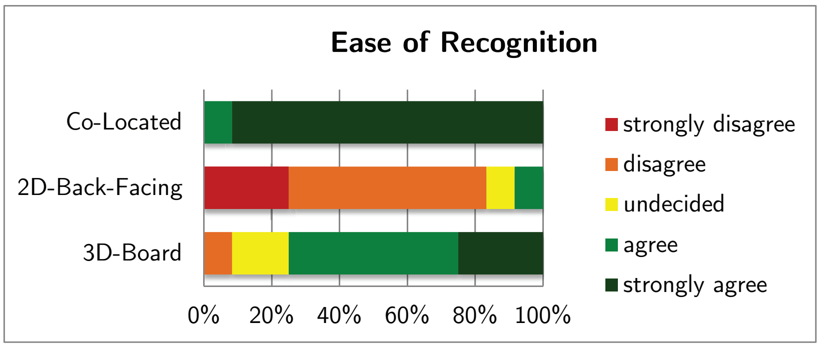
After each technique block, participants were asked to rate the different techniques from the instructor's as well as the observer's point of view. The rating was based on a 5-point Likert scale (1 = strongly agree; 5 = strongly disagree). Overall, 91.67% of the participants found the *3D-Board* technique either very easy or easy to use. This confirms that working with *3D-Board* is almost as intuitive as working in a *Co-Located* setup. In contrast, 75.0% of the participants rated *2D-Back-Facing* to be difficult or very difficult to utilize, as can be seen in Figure 5.9 (a). The outcome suggests that finding the correct posture as an instructor was very cumbersome for most participants, as discussed in the previous chapter. Very similar results were measured with the perceived accuracy of the techniques, visualized in Figure 5.9 (b). All participants agreed that *3D-Board* was accurate or very accurate, whereas 11 out of 12 or 91.67% disagreed when using the *2D-Back-Facing* technique. However, when asked if it was easy to recognize what the partner was referring to when using *3D-Board*, the agreement rate declined to 9 participants or 75.0%. While *3D-Board* is very accurate, the limited quality of the visualization can sometimes be cumbersome for the perception of the other user's actions. The hands of the instructor, for example, could be hard to recognize immediately if the skin color matched the color of the background or the cloth of a person. Participants had also trouble to interpret the partner's



(a)



(b)



(c)

Figure 5.9: The 5-point Likert scale ratings for the ease of use (a), the accuracy (b) and the ease of recognition of what the other user was referring to (c).

actions when using the *2D-Back-Facing* technique since important information is often being occluded. Thus, 10 out of 12 participants or 83,33% disagree or strongly disagree when using *2D-Back-Facing*, as also depicted in Figure 5.9 (c).

Figure 5.10 shows the task load ratings (1 = very low; 5 = very high) for all techniques, focusing on Physical Demand, Performance, Effort, and

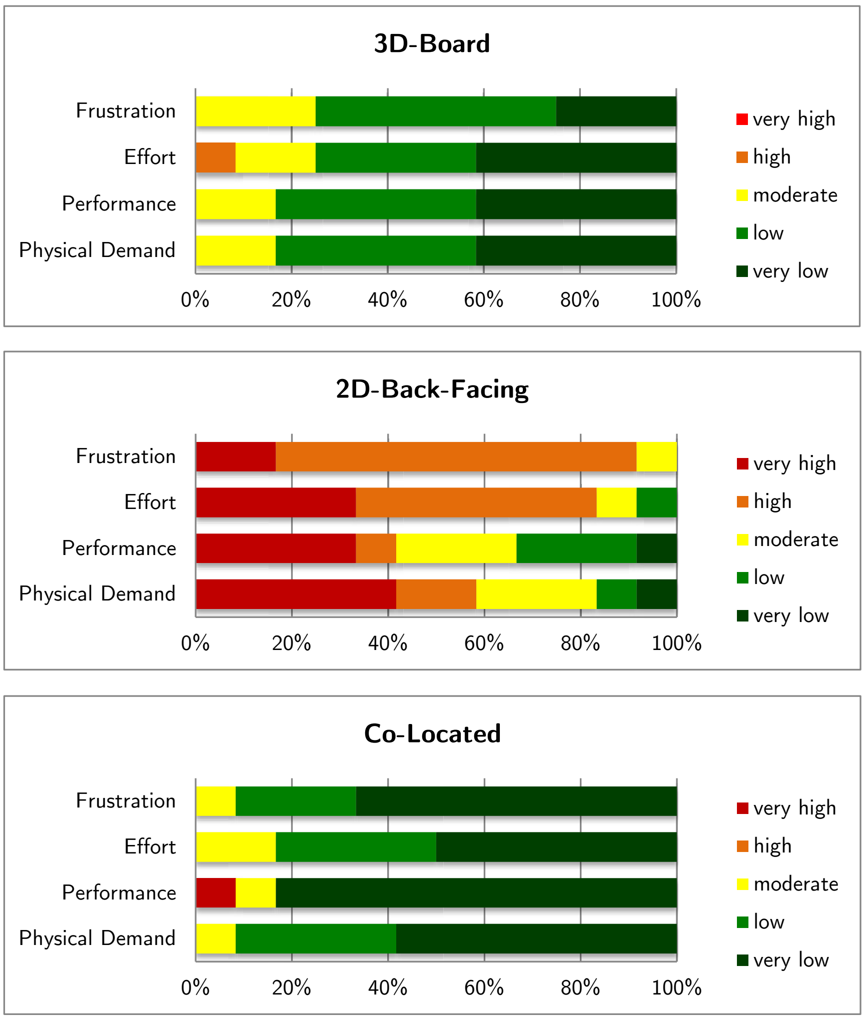


Figure 5.10: Task load ratings for the first experiment for *3D-Board*, *2D-Back-Facing*, and *Co-Located*, where significant differences could be found.

Frustration, where significant differences have been found. No significant main effects could be found for both Mental and Temporal Demand. The results show that *3D-Board* was almost as easy to use as when working co-located.

5.3 Experiment 2: Interior Design

While the first experiment was focused on accuracy and deictic gestures, the second experiment tested the performance of *3D-Board* in a more realistic scenario. The idea behind this experiment is to mainly get qualitative feed-

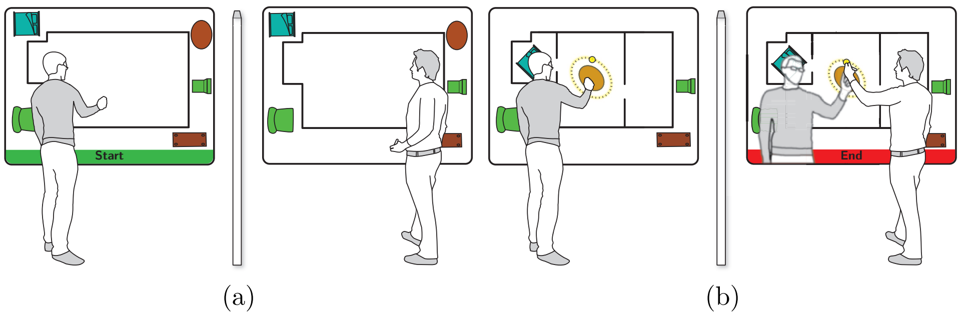


Figure 5.11: The operator received instructions (a) on how to draw inner walls and rearrange furniture on the floor plan with the intuitive tools (b).

back and impressions from the participants. However, in addition the time for each trial was logged.

5.3.1 Task

In the second task, the instructor was presented a printed out floor plan. The operator had to redraw the plan on an interactive whiteboard based on the orders of the instructor. The procedure, illustrated in Figure 5.11, is similar to the sequence of the first task:

1. Both participants were presented with the same empty floor plan, only the outer walls were given (cf. Figure 5.11 (a)). The furniture to place was arranged randomly at the outer border of the whiteboard. Only the instructor knew the final design of the plan.
2. After pressing the **Start** button to start the timer, the instructor showed the operator how to draw the inner walls and arrange the furniture (cf. Figure 5.11 (b)). The instructor was allowed to use both of his hands and voice commands to guide the operator. This should foster a natural interaction with extensive use of gestures.
3. The operator had to draw the missing inner walls with a simple pen tool. The furniture could be arranged by using intuitive pen gestures for translation, scale and rotation (cf. Figure 5.12).
4. Once the instructor was satisfied with the floor plan the trial was completed with the **End** button. In order to keep the completion time equal across all trial runs, the different floor plans consisted of varying arrangements of two pieces of furniture and two inner walls.

5.3.2 Procedure

A within-subject design was used. Per participant one trial run per technique had to be completed, resulting in a total of 36 trials:

	12	participants
×	3	Techniques (*3D-Board*, *2D-Back-Facing*, *Co-Located*)
×	1	task trials
	36	total trials.

For each of the six trial runs per group a unique floor plan was used. Since the three techniques (*3D-Board*, *2D-Back-Facing*, *Co-Located*) were counterbalanced, all techniques were tested equally often with the same floor plan, as shown in Table 5.2.

Before the trials began the participants could familiarized themselves with the tools. User feedback was collected through interviews and questionnaires, where participants had to rate their satisfaction, ease-of-use, and engagement. Task load ratings were collected using NASA TLX questionnaires. On average the second experiment lasted for 20 minutes per group.

5.3.3 Hypotheses

Since the second experiment is focused on evaluating the user experience, also the hypotheses target the preferences of the user.

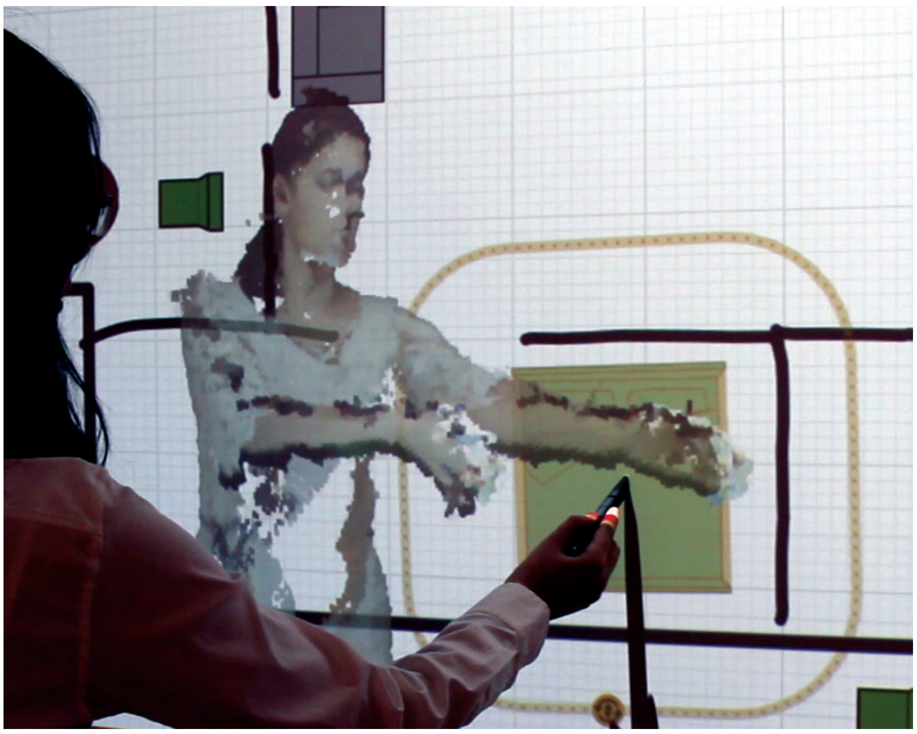


Figure 5.12: In the second experiment both participants had to cooperate to rearrange furniture on a floor plan. The yellow handle around a selected object was used to intuitively translate, rotate and scale the furniture.

Table 5.2: Counterbalancing of the independent variables *Technique* and *Floor Plan.*

Part.	Ind.Var	Sequence		
1 & 7	Technique	*Co-Located*	*2D-Back-Facing*	*3D-Board*
	Floor Plan	1	2	3
2 & 8	Technique	*2D-Back-Facing*	*3D-Board*	*Co-Located*
	Floor Plan	4	5	6
3 & 9	Technique	*3D-Board*	*Co-Located*	*2D-Back-Facing*
	Floor Plan	1	2	3
4 & 10	Technique	*Co-Located*	*2D-Back-Facing*	*3D-Board*
	Floor Plan	4	5	6
5 & 11	Technique	*2D-Back-Facing*	*3D-Board*	*Co-Located*
	Floor Plan	1	2	3
6 & 12	Technique	*3D-Board*	*Co-Located*	*2D-Back-Facing*
	Floor Plan	4	5	6

Hypothesis 4: *Co-Located is the preferred technique.*
The *Co-Located* technique was chosen as a baseline because it is the most natural way of interaction. Facial expressions and gestures, an integral part of close collaboration, will be clearest and thus the *Co-Located* technique will be the most preferred.

Hypothesis 5: *3D-Board raises workspace awareness.*
The front-facing visualization of *3D-Board* raises the awareness for the collaborator's actions. Important cues are obtained from the partner's facial expressions and gestures. This is an advantage over both the *2D-Back-Facing* as well as the *Co-Located* technique since awareness will be greater when facing someone directly.

Hypothesis 6: *The 3D-Board embodiment does not violate the collaborator's personal space.*
Users will not feel a violation of their personal space even when getting very close while working on the same data. This will also be true for the *2D-Back-Facing* technique but is a definite advantage over the *Co-Located* setup.

5.3.4 Quantitative Results

As in the first experiment, *3D-Board* ($M = 72.8\,\text{s}, SD = 19.0\,\text{s}$) was faster than the *2D-Back-Facing* technique ($M = 79.9\,\text{s}, SD = 25.7\,\text{s}$). The *Co-Located* ($M = 68.6\,\text{s}, SD = 23.4\,\text{s}$) technique was again quicker than *3D-Board*. However, the second experiment focused on gathering qualitative data and thus the trial count was too low to correctly measure time differences. Therefore, no statistical significance was given ($p = 1.0$) for all techniques.

5.3.5 Qualitative Results

Qualitative user feedback showed that participants were highly positive about the concept of *3D-Board*. Half of all participants preferred *3D-Board* over all other techniques, and no one of them mentioned *3D-Board* as their least favorite. Thus, *Hypothesis 4* has to be rejected since *3D-Board* and *Co-Located* were equally popular. On a five-point Likert scale (1 = very easy/strongly agree; 5 = very difficult/strongly disagree), participants found *3D-Board* easy to use (Median(ME) = 1.5) and easy to learn ($ME = 1.0$). In contrast, the *2D-Back-Facing* technique was rated as being moderate to use ($ME = 3.0$) and moderate to learn ($ME = 2.5$). In addition to that, 66.67% of the participants also felt a high or very high personal engagement when using *3D-Board*. The *Co-Located* scenario achieved better results with 83.33% while only 58.33% felt high or very high engagement when using *2D-Back-Facing*. A Wilcoxon signed-rank test showed significant differences for both usability ($z = 2.375, p = 0.018$) and learnability ($z = 2.326, p = 0.020$), for the comparison of *3D-Board* and *2D-Back-Facing*.

Figure 5.13 depicts the task load ratings (1 = very low; 5 = very high) for all techniques, focusing on Physical Demand, Performance, Effort, and Frustration, where significant differences have been found. As in the first experiment, no significant main effects could be found for both Mental and Temporal Demand. The post-condition questionnaire data (1 = very low; 5 = very high) were analyzed using two related samples Wilcoxon signed-rank tests. Overall, participants found *3D-Board* ($ME = 2.0$) less physically challenging than the 2D technique ($ME = 3.5$), $z = -2.414, p = 0.016$. In addition, most of the participants also felt the performance was better when using *3D-Board* ($ME = 4.5$) than with *2D-Back-Facing* ($ME = 4.0$), $z = -2.449, p = 0.014$. Also the effort was rated lower for *3D-Board* ($ME = 2.0$) than for *2D-Back-Facing* ($ME = 3.0$), with $z = -2.972, p = 0.003$. Finally, participant felt less frustration using the 3D technique ($ME = 2.0$) than using the 2D technique ($ME = 3.0$), $z = -2.739, p = 0.006$.

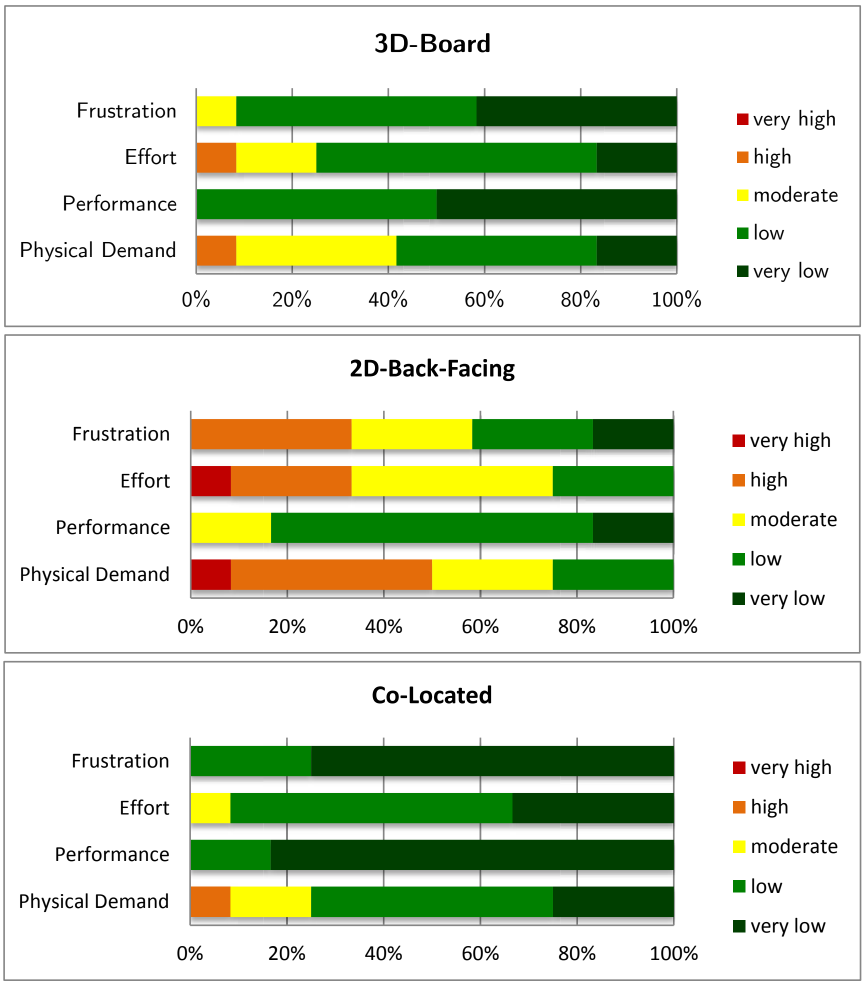


Figure 5.13: Task load ratings for the second experiment for *3D-Board*, *2D-Back-Facing*, and *Co-Located*, where significant differences could be found.

Awareness

Once asked about how easy it was to make oneself well understood as an instructor, the majority of the participants voted in favor of the front-facing *3D-Board* technique ($ME = 1.0$) compared to the back-facing technique ($ME = 2.0$), $z = 2.714, p = 0.007$ (cf. Figure 5.14 (a)). When acting as the operator, the majority of the participants knew better what the instructor was doing when using *3D-Board* ($ME = 1.5$) compared to *2D-Back-Facing* ($ME = 2.0$), $z = 2.810, p = 0.005$ (cf. Figure 5.14 (b)). These results suggest that participants had a harder time to make themselves understood with the orthogonal 2D projection. *3D-Board* on the other hand allowed for more natural gestures and it was easy to grasp the collaborator's intentions.

However, the *Co-Located* technique outperformed *3D-Board* ($ME = 1.0$) in terms of awareness with $z = -2.236, p = 0.025$. The root cause of this is the visual quality of *3D-Board*. When asking participants if they could clearly perceive the other user's facial expressions the 3D-visualization ($ME = 2.5$) achieved better results than the 2D-visualization ($ME = 3.0$) but without any significant difference, $z = 0.16, p = 0.873$ (cf. Figure 5.14 (c)). The *Co-Located* technique performed flawlessly ($ME = 1.0$), $z = 0.003, p = -3.017$. Thus, although *3D-Board* can raise the workspace awareness better than *2D-Back-Facing*, *Hypothesis 5* needs to be rejected. The *Co-Located* technique is raising the awareness better due to the rendering artifacts of *3D-Board*.

Territoriality

Similar to Doucette et al. [13], the participants were asked to rate their agreement with the statements, "I often had the feeling that my personal space was violated by my partner", and "I often had the feeling of invading my partner's personal space". While 75% of the participants disagreed or strongly disagreed with the first statement of being invaded by the partner under the *3D-Board* condition, only 58.33% did so under the *Co-Located* condition. For the second question, again 75% of the participants disagreed when using the 3D front-facing setup, while 66.67% of all participants disagreed in the *Co-Located* setup. In combination with the concluding interviews, discussed in the next section, this confirms *Hypothesis 6* since participants felt their private space less violated when working with *3D-Board*. However, a Wilcoxon signed-rank test showed no main effects.

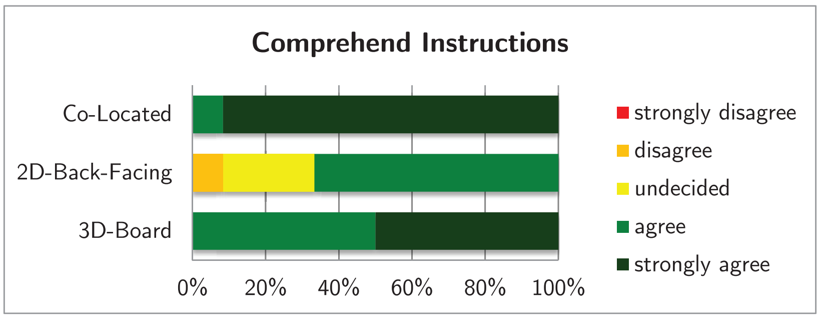
5.4 Interviews and Observations

Concluding interviews were conducted at the end of the study. Participants were asked to state the advantages and disadvantages of *3D-Board*. Including these interviews, the entire study resulted in a total duration of approximately 75 minutes per group.

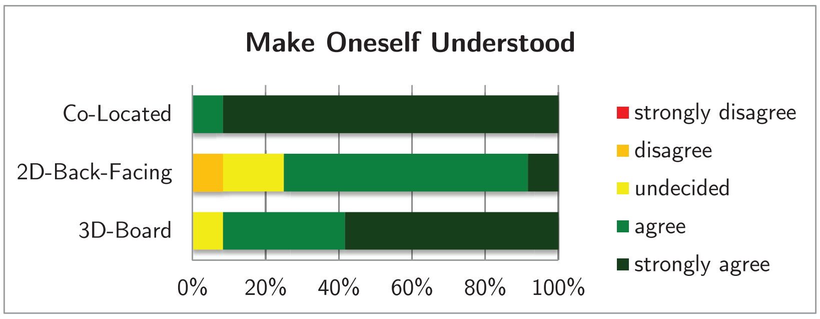
According to the participants, the major advantages of *3D-Board* were that the digital content was augmented with a virtual 3D embodiment. This leads to an intuitive gestural interaction and a direct connection between the digital content and the remote user.

> *While using* 3D-Board*, it was easy to see where the instructor was pointing to, because it gave me the impression that my partner was behind the wall. The interaction with someone who is behind the wall is exciting. – Participant 2A*

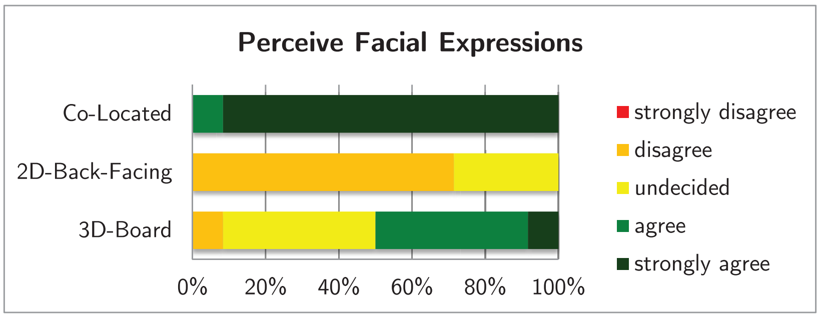
The face-to-face communication in life-size with the remote user made participants feel collaborating very closely with the remote user.



(a)



(b)



(c)

Figure 5.14: Awareness ratings for the ease of comprehending instructions (a), making oneself understood (b) and perceiving facial expressions (c).

> *I liked the feeling of communication in the* 3D-Board *condition, because the other person was looking at me, which made me feel more in contact with him. – Participant 5B*

Some of the participants also explicitly mentioned the advantage of not physically interfering with their partner, while working on the same content.

> *The collaboration was easier once you see the remote person in front of you - it gives you the feeling to work really face-to-face. And this, without stepping on someone's toes. – Participant 5A*

Observations showed increased gesticulation when using *3D-Board*. In the *Co-Located* setup users had less space and would thus act more defensively since they did not dare to cross the other user's workspace with their arms.

The disadvantage mentioned by most participants was the visual quality of *3D-Board*. While virtual embodiment was good enough to easily recognize gestures, participants often failed to read the other user's face.

> *I had a hard time perceiving the facial expressions. However, for the given tasks they were irrelevant and I only looked at my partner's gestures. – Participant 2A*

Another problem was the perception of spatial depth. It was hard to estimate the distance of the collaborator from the whiteboard, since spatial reference points were missing. In addition, for some participants it was sometimes hard to recognize the hand of the remote user since the transparency of the virtual embodiment made it hard to distinguish skin color from similar colored clothes.

The interviews and observations conclude that the possibility to collaborate face-to-face without any physical interference more than compensates for the inferior visual quality. Overall, participants expressed that they very much enjoyed working with *3D-Board*. It was the preferred technique for 6 out of 12 participants, as mentioned in the previous section.

Chapter 6

Conclusion and Future Work

This thesis presented *3D-Board*, a digital whiteboard featuring life-sized, virtual 3D embodiments of geographically distributed users. Capturing the whole body of the remote collaborator from the front increases the social presence and raises the awareness for gestures as well as gaze. Blending the visualization with the common workspace resolves the disparity between the representation of the distant user and the shared data. By tracking the observers head, the 3D embodiment appears to be standing behind the shared surface. Poses and deixis can be perceived and the teleimmersive impressions are enhance due to the motion parallax effects.

The facilitated intuitive interaction between the participants was evaluated in a usability assessment. The study showed the effectiveness of the proposed system. The compared *2D-Back-Facing* approach was outperformed in almost all categories. In certain situations *3D-Board* was even comparable to a co-located collaboration. This can be explained by the possibility of working simultaneously at the same spot without any physical interference, while co-located users kept a certain distance. Overall, the illusion of observing the remote cooperator standing behind the shared surface was pleasant for most participants and had a very high rate of acceptance.

However, there are many ways to improve the system, as the study results suggest. Users complained that the visual quality of the embodiment was often too low to fully grasp the facial expressions of the collaborator. In addition, the system cannot compensate very large holes in the rendering, caused when the arms are covering the body. The depth resolution of the Kinect sensor is inferior and the quad based visualization results in a low-resolution picture. Additionally, higher quality cameras, such as the Kinect v2, mounted at the top and bottom of the system could capture a cleaner image of the user. However, more cameras come at the cost of increased computational demands. While the point-based rendering is efficient, the extensive post-processing stage for filling holes is implemented on the CPU and therefore fairly slow. Hence, for the highest image quality, a parallelized or GPU-based implementation would be required to facilitate stable frame rates at high display resolutions with the used hardware system. If the system's performance is increased, the current implementation could be combined with the capabilities of the first prototype [65] to further enhance the rendering quality.

Future work could also include an evaluation of a collaboration between more than two whiteboards. Multiple virtual embodiments lead to an increased occlusion of the screen. In addition, multiple overlapping embodiments decrease the legibility and awareness for the other users. Thus, the visualization of the partner should be switched off on demand or the embodiment could change its transparency based on the distance to the whiteboard. The virtual representation is only of interest, if the user is working directly on the whiteboard. Stepping away from the screen should automatically decrease the transparency.

When multiple users are working simultaneously on the same screen, motion parallax enabled via the head tracking is not supported anymore. Whiteout the 3D effects the front-facing embodiment would still feature awareness for gestures and gaze but the hand pointing would be harder to perceive. This could be compensated by visualizing direct interaction with the whiteboard.

This thesis provided insights in the implementation of a novel telepresence system and the possibilities for further improvements are endless. *3D-Board* is nonetheless one small step towards intuitive remote collaboration.

Appendix A

Sourcecode

A.1 Vertex Shader

```
struct GeoShaderInput { };

GeoShaderInput VertexShader()
{
    return (GeoShaderInput)0;
}
```

A.2 Geometry Shader

```
//--------------------------------------------
// Structures
//--------------------------------------------
struct GeoShaderInput { };

struct PixelShaderInput { float4 Pos:SV_POSITION; float4 Col:COLOR; };

//--------------------------------------------
// Textures
//--------------------------------------------
Texture2D<int>    depthTex   : register(t0);
Texture2D<float4> colorTex   : register(t1);
Texture2D<int>    depthRefTex   : register(t2);
SamplerState      colorSampler : register(s0);

//--------------------------------------------
// Constant Buffer Variables
//--------------------------------------------
cbuffer cbChangesEveryFrame : register(b0)
{
  matrix instrinsicsInverse;
  matrix extrinsicsInverse;
  matrix viewMatrix;
  matrix projectionMatrix;
};
```

```

//-------------------------------------------
// Constants
//-------------------------------------------
static const int    depthWidth = 640;
static const int    depthHeight = 480;
static const float  quadSize = 2.5f;
static const float4  quadScale = float4(1.0/depthWidth, 1.0/depthHeight,
     0.0, 0.0) * quadSize;

// vertex offsets for building a quad from a depth pixel
static float4 quadOffsets[4] =
{
    float4(-0.5f, -0.5f, 0.0f, 0.0f),
    float4( 0.5f, -0.5f, 0.0f, 0.0f),
    float4(-0.5f,  0.5f, 0.0f, 0.0f),
    float4( 0.5f,  0.5f, 0.0f, 0.0f)
};

//-------------------------------------------
// Functions
//-------------------------------------------

bool invalidPixel(float depth, float depthRef)
{
  // use the minimum of near mode
  static const float minDepth = 0.3f;
    // use the maximum of standard mode
  static const float maxDepth = 4.0f;
  // difference in meters between reference frame and current frame
  static const float maxDepthDiffTheta = 0.5f;
  // check that depth is in the valid range
  if (depth < minDepth || depth > maxDepth || abs(depth - depthRef) <
    maxDepthDiffTheta)
  {
    return true;
  }

  return false;
}

void ExtractQuad(float4 viewPos, float4 viewspaceScale, inout
     TriangleStream<PixelShaderInput> triStream, PixelShaderInput output)
{
  // expand the current point into four vertices
  [unroll]
  for (uint i = 0; i < 4; ++i)
  {
    // expand quad in view space - it always faces the camera
    float4 viewPosExpanded = viewPos + quadOffsets[i] * viewspaceScale;
    // project vertex onto virtual camera
    output.Pos = mul(viewPosExpanded, projectionMatrix);
    // append vertex to output triangle stream
    triStream.Append(output);
  }
}

//-------------------------------------------
// Geometry Shader
//
// takes single vertex and expands it into the 4 vertices of a quad;
// depth sampled from Kinect's depth texture mapped to color space;
// color sampled from Kinect's color texture;
//-------------------------------------------
[maxvertexcount(4)]
void GeometryShader(point GeoShaderInput vertex[1], uint primID :
    SV_PrimitiveID, inout TriangleStream<PixelShaderInput> triStream)
{
  // sample depth from texture using the current pixel index primID
  int3 texCoord = int3(primID % depthWidth, primID / depthWidth, 0);
  // depth from texture in millimeters - converted to meters
  float depth = depthTex.Load(texCoord) / 1000.0f;
  float depthRef = depthRefTex.Load(texCoord) / 1000.0f;

  // background subtraction
  if (invalidPixel(depth, depthRef)) { return; }

  // project from image to camera space using the intrinsic matrix
  float4 imagePos = float4(texCoord.xy * depth, depth, 1.0f);
  float4 cameraPos = mul(imagePos, instrinsicsInverse);

  // project from camera to world space using the extrinsic matrix
  float4 worldPos = mul(cameraPos, extrinsicsInverse);

  // transform from world space to space of virtual, head tracked camera
  float4 viewPos = mul(worldPos, viewMatrix);

  // expand quad based on depth - points farther away will scale up
  float4 viewspaceScale = quadScale * depth;

  // output variable holding the quad
  PixelShaderInput output;

  // base color texture sample lookup coords, in [0,1]
  float2 depthWidthHeight = float2(depthWidth, depthHeight);
  float2 colorTexCoords = texCoord.xy/depthWidthHeight;
  // sample the color texture
  output.Col = colorTex.SampleLevel(colorSampler, colorTexCoords, 0);

  // extract a fixed-size quad
  ExtractQuad(viewPos, viewspaceScale, triStream, output);
}
```

A.3 Pixel Shader

```
struct PixelShaderInput { float4 Pos:SV_POSITION; float4 Col:COLOR; };

```

```
float4 PixelShader(PixelShaderInput input) : SV_Target
{
    return float4(input.Col.rgb, 1.0);
}
```

Appendix B

CD Content

Format: CD-ROM, Single Layer, ISO9660-Format

B.1 PDF-Dateien

Pfad: /

3D-Board_JakobZillner.pdf .	Master Thesis

B.2 Source Code

Pfad: /source_code/

3DBoard/	source code of *3D-Board*
EvaluationApplications/ . . .	source code of experiments

B.3 Study Data

Pfad: /study_data/

evaluation_data/	collected evaluation data
questionnaires/	evaluation questionnaires

B.4 Images

Pfad: /images/

1_introduction/	figures used in Chapter 1
2_inspiration_concepts/ . .	figures used in Chapter 2
3_implementation/	figures used in Chapter 3
4_demo_applications/ . . .	figures used in Chapter 4
5_evaluation/	figures used in Chapter 5

References

Literature

[1] Stephan Beck et al. “Immersive Group-to-Group Telepresence”. In: *IEEE Transactions on Visualization and Computer Graphics* 19.4 (2013), pp. 616–625 (cit. on pp. 8, 9).

[2] Steve Benford et al. “User Embodiment in Collaborative Virtual Environments”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* Denver, Colorado, USA: ACM Press/Addison-Wesley Publishing Co., 1995, pp. 242–249 (cit. on p. 5).

[3] Sara A. Bly, Steve R. Harrison, and Susan Irwin. “Media Spaces: Bringing People Together in a Video, Audio, and Computing Environment”. In: *Communications of the ACM* 36.1 (1993), pp. 28–46 (cit. on p. 6).

[4] Greg Borenstein. *Making Things See.* O’Reilly Media, Inc., 2012 (cit. on p. 3).

[5] Erin Bradner and Gloria Mark. “Social Presence with Video and Application Sharing”. In: *GROUP ’01.* Boulder, Colorado, USA: ACM, 2001, pp. 154–161 (cit. on p. 5).

[6] Gary Bradski and Adrian Kaehler. *Learning OpenCV.* O’Reilly, 2008 (cit. on pp. 15, 18–21, 24, 28).

[7] Duane C. Brown. “Close-range camera calibration”. In: *Photogrammetric Engineering* 37.8 (1971), pp. 855–866 (cit. on p. 19).

[8] Wilhelm Burger and Burge Mark J. *Principles of Digital Image Processing – Advanced Methods.* Vol. 3. Springer Undergraduate Topics in Computer Science, 2013 (cit. on p. 29).

[9] Wilhelm Burger and Burge Mark J. *Principles of Digital Image Processing – Fundamental Techniques.* Vol. 1. Springer Undergraduate Topics in Computer Science, 2009 (cit. on p. 29).

[10] D. Alex Butler et al. “Shake’N’Sense: Reducing Interference for Overlapping Structured Light Depth Cameras”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Austin, Texas, USA: ACM, 2012, pp. 1933–1936 (cit. on p. 3).

[11] Fu Chang, Chun-Jen Chen, and Chi-Jen Lu. “A Linear-time Component-labeling Algorithm Using Contour Tracing Technique”. In: *Computer Vision and Image Understanding* 93.2 (2004), pp. 206–220 (cit. on p. 39).

[12] Li Chen, Hui Lin, and Shutao Li. “Depth image enhancement for Kinect using region growing and bilateral filter”. In: *International Conference on Pattern Recognition*. Tsukuba, Japan: IEEE, 2012, pp. 3070–3073 (cit. on p. 29).

[13] Andre Doucette et al. “Sometimes when we touch: how arm embodiments change reaching and collaboration on digital tables”. In: *Proceedings of the Conference on Computer Supported Cooperative Work*. San Antonio, Texas, USA: ACM, 2013, pp. 193–202 (cit. on pp. 6, 68).

[14] Jörg Edelmann et al. “Face2Face – A System for Multi-Touch Collaboration With Telepresence”. In: *International Conference on Emerging Signal Processing Applications*. Las Vegas, NV, USA: IEEE, 2012, pp. 159–162 (cit. on p. 8).

[15] Rob van Eijk et al. “Human sensitivity to eye contact in 2D and 3D videoconferencing”. In: *Proceedings of the International Workshop on Quality of Multimedia Experience*. IEEE, 2010, pp. 76–81 (cit. on pp. 1, 5).

[16] Andy Field. *Discovering Statistics using IBM SPSS Statistics*. 4th ed. Sage Publications Ltd., 2013 (cit. on p. 58).

[17] M. Forte and G. Kurillo. “Cyberarchaeology: Experimenting with Teleimmersive Archaeology”. In: *International Conference on Virtual Systems and Multimedia*. Seoul, Korea: IEEE, 2010, pp. 155–162 (cit. on p. 8).

[18] Susan R. Fussell et al. “Gestures over Video Streams to Support Remote Collaboration on Physical Tasks”. In: *International Journal of Human-Computer Interaction* 19.3 (2004), pp. 273–309 (cit. on p. 5).

[19] Aaron M. Genest et al. “KinectArms: A Toolkit for Capturing and Displaying Arm Embodiments in Distributed Tabletop Groupware”. In: *Proceedings of the Conference on Computer Supported Cooperative Work*. San Antonio, Texas, USA: ACM, 2013, pp. 157–166 (cit. on p. 6).

[20] Aaron Genest and Carl Gutwin. “Characterizing Deixis over Surfaces to Improve Remote Embodiments”. In: *Proceedings of the European Conference on Computer Supported Cooperative Work*. Aarhus, Denmark: Springer London, 2011, pp. 253–272 (cit. on p. 5).

[21] Tovi Grossman, Patrick Baudisch, and Ken Hinckley. “Handle Flags: Efficient and Flexible Selections for Inking Applications”. In: *Proceedings of Graphical Interfaces*. Kelowna, British Columbia, Canada: Canadian Information Processing Society, 2009, pp. 167–174 (cit. on p. 54).

[22] Raja Gumienny et al. “Tele-Board: Enabling Efficient Collaboration In Digital Design Spaces”. In: *Conference on Computer Supported Cooperative Work in Design*. Lausanne, Switzerland: IEEE, 2011, pp. 47–54 (cit. on pp. 6, 9).

[23] Carl Gutwin and Saul Greenberg. “A Descriptive Framework of Workspace Awareness for Real-Time Groupware”. In: *Computer Supported Cooperative Work* 11.3-4 (2002), pp. 411–446 (cit. on p. 1).

[24] Carl Gutwin and Greenberg Saul. “The Importance of Awareness for Team Cognition in Distributed Collaboration”. In: *Team Cognition: Understanding the Factors That Drive Process and Performance*. American Psychological Association Press, 2004, pp. 177–203 (cit. on pp. 1, 5, 9).

[25] Edward T. Hall. *The Hidden Dimension*. Anchor Books, 1990 (cit. on p. 51).

[26] Michael Haller et al. “The NiCE Discussion Room: Integrating Paper and Digital Media to Support Co-Located Group Meetings”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Atlanta, Georgia, USA: ACM, 2010, pp. 609–618 (cit. on p. 12).

[27] Richard Hartley and Andrew Zisserman. *Multiple View Geometry*. Cambridge University Press, 2004 (cit. on p. 20).

[28] David A. Harville. *Matrix Algebra From a Statistician’s Perspective*. Springer, 2008 (cit. on p. 22).

[29] Jörg Hauber et al. “Spatiality in Videoconferencing: Trade-offs Between Efficiency and Social Presence”. In: *Proceedings of the Conference on Computer Supported Cooperative Work*. Banff, Alberta, Canada: ACM, 2006, pp. 413–422 (cit. on pp. 1, 6, 9).

[30] Adrian Ilie and Greg Welch. “Ensuring Color Consistency Across Multiple Cameras”. In: *International Conference on Computer Vision*. Beijing, China: IEEE, 2005, pp. 1268–1275 (cit. on p. 28).

[31] Hiroshi Ishii and Minoru Kobayashi. “ClearBoard: A Seamless Medium for Shared Drawing and Conversation with Eye Contact”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Monterey, California, USA: ACM, 1992, pp. 525–532 (cit. on pp. 6, 9).

[32] Shahram Izadi et al. “C-Slate: A Multi-Touch and Object Recognition System for Remote Collaboration using Horizontal Surfaces”. In: *Tabletop – International Workshop on Horizontal Interactive Human-Computer Systems*. Newport, Rhode Island, USA: IEEE, 2007, pp. 3–10 (cit. on p. 6).

[33] Ramesh Jain, Rangachar Kasturi, and Brian G. Schnuck. *Machine Vision*. McGraw-Hill, 1995 (cit. on p. 18).

[34] Ricardo Jota et al. “A Comparison of Ray Pointing Techniques for Very Large Displays”. In: *Proceedings of Graphics Interface*. Ottawa, Ontario, Canada: Canadian Information Processing Society, 2010, pp. 269–276 (cit. on p. 5).

[35] Sasa Junuzovic et al. “IllumiShare: Sharing Any Surface”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Austin, Texas, USA: ACM, 2012, pp. 1919–1928 (cit. on p. 6).

[36] Bernhard Kainz et al. “OmniKinect: Real-time Dense Volumetric Data Acquisition and Applications”. In: *Symposium on Virtual Reality Software and Technology*. Toronto, Ontario, Canada: ACM, 2012, pp. 25–32 (cit. on p. 24).

[37] Kibum Kim et al. “TeleHuman: Effects of 3D Perspective on Gaze and Pose Estimation with a Life-size Cylindrical Telepresence Pod”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Austin, Texas, USA: ACM, 2012, pp. 2531–2540 (cit. on pp. 7, 10).

[38] David Kirk, Tom Rodden, and Danaë Stanton Fraser. “Turn It This Way: Grounding Collaborative Action with Remote Gestures”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. San Jose, California, USA: ACM, 2007, pp. 1039–1048 (cit. on p. 1).

[39] Jesper Kjeldskov et al. “Eye Contact over Video”. In: *SIGCHI Extended Abstracts on Human Factors in Computing Systems*. Toronto, Canada: ACM, 2014, pp. 1561–1566 (cit. on p. 7).

[40] Johannes Kopf et al. “Joint Bilateral Upsampling”. In: *ACM Transactions on Graphics* 26.3 (2007) (cit. on p. 29).

[41] A. Kunz, T. Nescher, and M. Küchler. "CollaBoard: A Novel Interactive Electronic Whiteboard for Remote Collaboration with People on Content". In: *International Conference on Cyberworlds*. Singapore: IEEE, 2010, pp. 430–437 (cit. on pp. 7, 9, 52).

[42] David Ledo et al. "OneSpace: Shared Depth-corrected Video Interaction". In: *SIGCHI Extended Abstracts on Human Factors in Computing Systems*. Paris, France: ACM, 2013, pp. 997–1002 (cit. on p. 8).

[43] Jiannan Li et al. "Interactive Two-sided Transparent Displays: Designing for Collaboration". In: *Conference on Designing Interactive Systems*. Vancouver, BC, Canada: ACM, 2014, pp. 395–404 (cit. on p. 7).

[44] Frank Luna. *Introduction to 3D Game Programming with DirectX 11*. Mercury Learning Information, 2012 (cit. on p. 35).

[45] Yi Ma et al. *An Invitation to 3-D Vision: From Images to Geometric Models*. Springer, 2004 (cit. on p. 17).

[46] Andrew Maimone and Henry Fuchs. "Encumbrance-Free Telepresence System with Real-Time 3D Capture and Display using Commodity Depth Cameras". In: *International Symposium on Mixed and Augmented Reality*. Basel, Switzerland: IEEE, 2011, pp. 137–146 (cit. on pp. 8, 28).

[47] Andrew Maimone et al. "Enhanced Personal Autostereoscopic Telepresence System using Commodity Depth Cameras". In: *Computers & Graphics* 36.7 (2012), pp. 791–807 (cit. on pp. 24, 38).

[48] Klara Nahrstedt. "3D Teleimmersion for Remote Injury Assessment". In: *International Conference on Security and Management*. Las Vegas, USA: ACM, 2012, pp. 21–24 (cit. on p. 8).

[49] Anthony J. Pettofrezzo. *Matrices and Transformations*. 1st ed. Dover Publications Inc., 1978 (cit. on p. 22).

[50] David Pinelle et al. "The Effects of Co-present Embodiments on Awareness and Collaboration in Tabletop Groupware". In: *Proceedings of Graphics Interface*. Windsor, Ontario, Canada: Canadian Information Processing Society, 2008, pp. 1–8 (cit. on pp. 1, 6).

[51] Szymon Rusinkiewicz and Marc Levoy. "Efficient variants of the ICP algorithm". In: *International Conference on 3-D Digital Imaging and Modeling*. Quebec, Canada: IEEE, 2001, pp. 145–152 (cit. on p. 25).

[52] Jamie Shotton et al. "Real-time Human Pose Recognition in Parts from Single Depth Images". In: *Proceedings of the Conference on Computer Vision and Pattern Recognition*. Colorado Springs, USA: IEEE, 2011, pp. 1297–1304 (cit. on p. 3).

[53] Jonathan Steuer. “Defining Virtual Reality: Dimensions Determining Telepresence”. In: *Communication in the Age of Virtual Reality*. L. Erlbaum Associates Inc., 1995, pp. 33–56 (cit. on p. 1).

[54] Kar-Han Tan et al. “ConnectBoard: Enabling Genuine Eye Contact and Accurate Gaze in Remote Collaboration”. In: *IEEE Transactions on Multimedia* 13.3 (2011), pp. 466–473 (cit. on pp. 7, 9).

[55] Anthony Tang, Michael Boyle, and Saul Greenberg. “Display and Presence Disparity in Mixed Presence Groupware”. In: *Proceedings of the Conference on Australasian User Interface*. Dunedin, New Zealand: Australian Computer Society, Inc., 2004, pp. 73–82 (cit. on p. 1).

[56] Anthony Tang, Carman Neustaedter, and Greenberg Saul. “VideoArms: Embodiments for Mixed Presence Groupware”. In: *People and Computers XX – Engage (Proceedings of HCI 2006)*. IEEE Computer Society: Springer, 2006, pp. 85–102 (cit. on p. 1).

[57] John C. Tang and Scott Minneman. “VideoWhiteboard: Video Shadows to Support Remote Collaboration”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New Orleans, Louisiana, USA: ACM, 1991, pp. 315–322 (cit. on p. 6).

[58] Alexandru Telea. “An Image Inpainting Technique Based on the Fast Marching Method”. In: *Journal Graphics, GPU & Game Tools* 9.1 (2004), pp. 23–34 (cit. on p. 40).

[59] Philip Tuddenham and Peter Robinson. “Territorial Coordination and Workspace Awareness in Remote Tabletop Collaboration”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Boston, MA, USA: ACM, 2009, pp. 2139–2148 (cit. on p. 6).

[60] R. Vasudevan et al. “High-Quality Visualization for Geographically Distributed 3-D Teleimmersive Applications”. In: *IEEE Transactions on Multimedia* 13.3 (2011), pp. 573–584 (cit. on p. 8).

[61] James M. van Verth and Lars M. Bishop. *Essential Mathematics for Games and Interactive Applications: A Programmer’s Guide*. 2nd ed. Taylor & Francis Ltd., 2008 (cit. on p. 24).

[62] Jun Xie et al. “Fine registration of 3D point clouds with iterative closest point using an RGB-D camera”. In: *International Symposium on Circuits and Systems*. Beijing, China: IEEE, 2013, pp. 2904–2907 (cit. on p. 25).

[63] Song Zhang. “High-resolution, Real-time 3-D Shape Measurement”. PhD thesis. New York, USA: Stony Brook University, 2005 (cit. on p. 3).

[64] Zhengyou Zhang. “A Flexible New Technique for Camera Calibration”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11 (2000), pp. 1330–1334 (cit. on p. 19).

[65] Jakob Zillner et al. “3D-Board: A Whole-body Remote Collaborative Whiteboard”. In: *Symposium on User Interface Software and Technology*. Honululu, USA: ACM, 2014, to appear (cit. on pp. 2, 51, 72).

Online sources

[66] *A Day Made of Glass Extended Montage*. 2013. URL: https://www.youtube.com/watch?v=PfgmlVxLC9w&feature=c4-overview-vl&list=PL363989F7BCF53A36 (cit. on p. 7).

[67] Robert Kooima. *Generalized Perspective Projection*. 2009. URL: http://csc.lsu.edu/~kooima/pdfs/gen-perspective.pdf (visited on 07/01/2014) (cit. on pp. 38, 39).