Advanced Interaction Techniques for Remote Collaboration Systems Using an RGBD Camera

Barbara J. Schwankl

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im September 2015

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 18, 2015

Barbara J. Schwankl

Contents

Vorwort

Fortschreitende Technologie ermöglicht es uns, länder- und kontinentübergreifend miteinander zu kommunizieren. Jedoch hat sich gezeigt, dass die Kommunikation via Email und Smartphone auch den persönlichen Kontakt vernachlässigen kann. Diese Arbeit fokussiert daher Videokonferenzen, welche persönliche Besprechungen imitieren und somit die persönliche Note mit einfließen lassen. Großen Dank an meinen Betreuer Dr. Michael Haller, welcher mir mit Expertise zur Seite stand und mich mit seiner konstruktiven Kritik ständig forderte, meine Arbeit noch zu verbessern. Danke auch an Dr. Wilhelm Burger für seine Hilfe bei der Leistungsverbesserung meiner Algorithmen.

Mit dieser Arbeit endet ein langjähriger, erfahrungsreicher Abschnitt in meinem Leben - das Studium. Ich freue mich schon darauf, mit all den gewonnenen wertvollen Erfahrungen mit voller Kraft voraus in den nächsten Abschnitt zu starten. Die Zeit des Studiums war nicht immer leicht, und ohne die Unterstützung einiger Personen wäre ein erfolgreicher Abschluss nicht möglich gewesen. Aus tiefstem Herzen danke ich Martin für seine endlose Liebe, seinen immerwährenden Rückhalt und nicht zuletzt seinem unermüdlichen Korrekturlesen dieser Arbeit. Besonderer Dank gilt meinen Eltern, die mir dieses Studium ermöglicht und über all die Jahre an mich geglaubt haben. Lieben Dank an meine Großeltern, Schwiegereltern und Geschwister für euer Mitfiebern und eure offenen Ohren. Danke David für deine sorgfältige Korrekturlesung. Danke auch an meine Freunde und Studienkollegen, für gesellige Stunden, hitzige Diskussionen und nächtelange, intensive Treffen zum Ausarbeiten gemeinsamer Übungen. In manchen von ihnen habe ich Freunde fürs Leben gefunden.

Abstract

This work presents an enhanced system supporting remote collaboration using a digital whiteboard and an RGBD camera. Showing the remote partner behind the transparent collaboration surface imitates a face-to-face meeting, revealing significant mimics, gestures and interactivity of the partner. Thus, a personalized feeling for the users is created. Three different gesture visualization scenarios are introduced in order to compensate for the perspective distortion that is due to the camera positioning. Several seed fill algorithms are compared to increase the performance of the system, considering the amount of data that has to be processed to render the motion images. A prototype has been developed to investigate the relevance and adaptability of the system within a workaday life.

Kurzfassung

Diese Arbeit beschäftigt sich mit einem Kollaborationssystem zur Ergänzung von Videokonferenzsystemen, bei welchem ein digitales Whiteboard zur Interaktion und eine RGBD Kamera als Sensor verwendet werden. Ziel ist es, ein System zur Verfügung zu stellen, bei welchem der Gesprächspartner scheinbar hinter dem transparenten Whiteboard steht. Dadurch kann direkt und persönlich mit dem Gesprächspartner kommuniziert werden, wobei zusätzlich zur Sprache auch Mimik und Gestik zu besserer Zusammenarbeit und einer erfolgreichen Kommunikation beitragen. Diese zusätzlichen Eindrücke auf der Meta-Ebene ermöglichen den Aufbau einer tieferen Verbundenheit zum Kommunikationspartner, als es über eine reine auf Ton basierende Kommunikation möglich wäre. In einem Prototypen wurden drei verschiedene Varianten für die Visualisierung des Gesprächspartners verwirklicht, um die perspektivischen Verzerrungen, welche durch die Positionierung der Kamera verursacht werden, zu kompensieren. Zusätzlich wurden effiziente Algorithmen verwendet, um das System in einem herkömmlichen Büro ohne zusätzliche Aufwände einsetzen zu können.

Chapter 1

Introduction

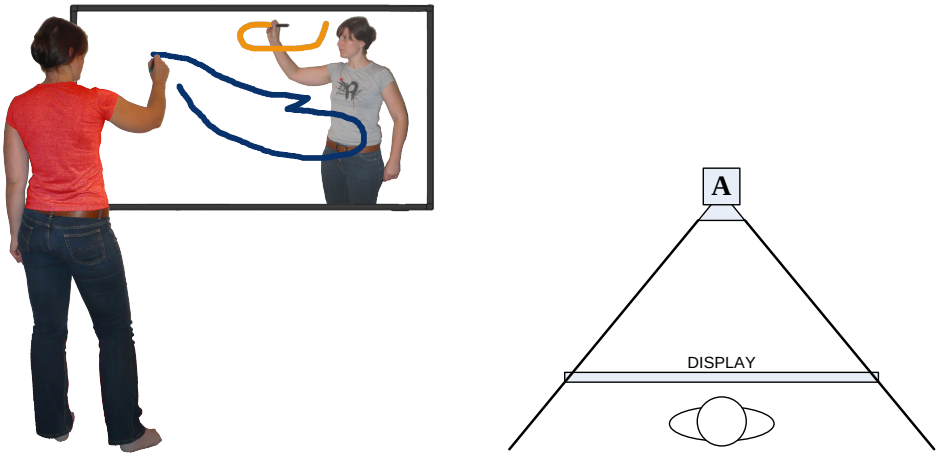
Teleconferencing technologies enable users to work collaboratively without the need to share the same room. Collaborators may come from different cities or different countries, have a different mother-language and a different cultural background. Therefore, it is important that remote collaboration concepts set an increased focus on communication and interaction. Video supported collaboration offers multiple advantages as the user can transport information through gestures and mimics without interrupting the currently talking user [7]. Previous studies showed that in video conferencing systems, body language availability and the field-of-view of the transmitted data influences the users' communicative attitude [16, 17]. The subjective feeling of dominance within a conversation depends on the users' gender. Men feel more dominant using body language while women feel more dominant with their body language not being transmitted. Furthermore, eye-contact and gaze awareness are essential to conceive meta-information [2]. The conceived distance to the communication partner influences how comfortable the participants feel. Hall [3] describes these so-called proxemics in his work. Digital collaboration tools, such as interactive whiteboards, can break the barriers of proxemics [21].

The main aim of this thesis is to find a way to imitate face-to-face collaboration for remote participants and enhance gesture visualizations with a simple hardware setup.

1.1 Motivation

The ideal scenario for a remote collaboration tool is a digital screen on which the collaborative content is shared and edited by all users simultaneously (cf. Fig. 1.1a). On the screen and behind the shared content, the remote collaborator is seen like through a transparent surface. Figure 1.1b shows a setup that endorses this vision. However, there are several drawbacks linked to that approach. The camera that is placed behind the screen captures the

user and the shared content. Similar as described by Ishii et al. [8], written content appears mirrored, therefore it would have to be excluded from the video stream first, then mirrored, and subsequently combined with the remote user's image again. Furthermore, this setup requires an unreasonable



(a) Remote collaboration (b) Ideal camera position

Figure 1.1: The remote collaboration scenario. The user writes on a collaborative surface and sees the collaborator behind the shared content (a). The ideal camera placement would be behind a transparent collaborative wall that captures the user from the front. In this way, a face-to-face communication is imitated (b).

amount of space for conventional offices. The screen has to be set up in the middle of the room to leave enough space to place the camera behind. Thereby, a suitable distance from the camera to the screen is required to be able to capture the user when standing still as well as on moving. A prototype was created that surpasses the weaknesses of the approaches presented in Chapter 2. The demands for the system are as follows:

- **Global availability and affordability.** A fundamental idea behind elaborated remote collaboration is availability of the system for all participating users, disregarding their business site location and their financial capabilities. Remote collaboration often includes people from foreign countries, who may encounter difficulties to access the latest technology.
- **Maximum visible user representation.** A maximum of the remote user's body with a focus on the mimics and gestures should be perceived without occluding the collaborative content.
- **Transferability to other users.** The system should be applicable to different people, not being customized for one person, but rather be

suitable for multiple users with different profiles, habits and different body heights.

Multiuser support is an essential requirement for collaborative environments. The prospective system requires support for multiple users within a conference, one user per system setup. All collaborators should use a common surface to sketch, draw and write down their desired content. Indicating each user with a personalized mouse cursor is often not sufficient to quickly link the writing user with the written content, especially if several users produce content at the same time. Therefore, a focus shall be put on the linkage between user and content. Transmitting the whole body of a user including the writing hand helps to understand the relation between user and writing. Opposed to mouse cursors, projecting users behind the collaborative surface imitates a face-to-face communication and creates an interpersonal relationship. Furthermore, using human body parts to point helps observing the remote collaborator and the shared content simultaneously. Additionally, the content may be emphasized through the remote user's mimics and gestures.

In order to to allow the usage in conventional offices, it is required that the camera is placed next to the interactive wall. However, this camera placement implicates a problem – the user is not captured from the front. Therefore, the users' remote representation has to be manipulated in a way that natural interaction with the other collaborators is enabled. This thesis presents several approaches to solve this problem.

1.2 Outline

This thesis covers all stages within a remote collaboration pipeline and presents the used algorithms in detail. Chapter 2 discusses related work and state of the art technology. Chapter 3 introduces the concept and the setup for a prototype aiming to enhances remote collaboration, and all components of the used RGBD sensor and their limitations are explained. The remote collaboration pipeline is covered in Chapter 4. The data acquisition, frame transformation and rendering stages are described in all its detail and the used algorithms for the prototype are elaborated. Furthermore, problems and limitations within the pipeline are discussed. Finally, future work and prospective enhancements on the prototype are outlined in Chapter 5.

Chapter 2

Related work

2.1 State of the art

Tan et al. [15] present *ConnectBoard*, a see-through display with a camera and a projector mounted behind the screen respectively. The screen shows the shared media and the remote user in the background while the camera captures the local user (cf. Fig. 2.1). Subsequently, the image of the local user is transmitted to the remote user. Genuine eye-contact is achieved using face detectors by automatically shifting the image of the remote user onto the path between the local user and the camera. In this way, eye-contact is always given between two users even in case of movement.

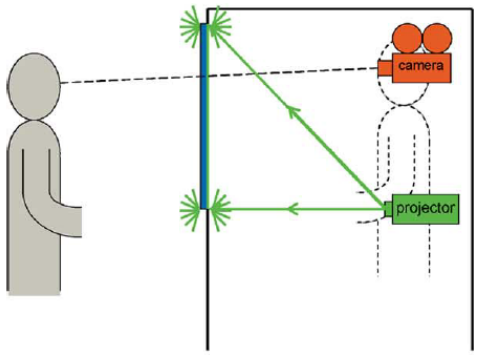


Figure 2.1: The setup of *ConnectBoard* [15]. The projector (green) projects the assembled image of the remote user with the shared content onto the screen. The camera (red) captures the local user and transmits the image back to the remote user. The transmitted image is mirrored for correct presentation.

This approach is similar to the work of Ishii and Kobayashi [8] who introduced *ClearBoard* They also used a transparent screen with a projector and a camera in the back to provide gaze awareness.

Izadi et al. [9] introduced a collaborative system that enables users to share and exchange information in public spaces. Their approach requires large displays and commonly accessible mice and keyboards to interact with the system. Data is transferred via USB flash drives. Each flash drive is mapped to a user that gains an individual space on the public surface. Users exchange data by simply dragging a file onto the other users' avatar (cf. Fig. 2.2).

Figure 2.2: *Dynamo* offers the users a platform to share and exchange content [9]. Furthermore, users can decide themselves what kind of content to share with whom.

IllumiRoom and *RoomAlive* are proof-of-concept prototypes developed by Jones et al. [11, 12] that augment parts of rooms. An ultra-wide field of view projector is used to display content and a depth camera is used to recognize object structures. According to the different structures, content is rendered differently (cf. Fig. 2.3). *RoomAlive* tracks the bodies of the participants and dynamically adapts the content according to their behavior. Intentionally designed for gaming, these systems can also be adopted to virtual conferences.

Similar to these approaches, Kasahara et al. [13] present *Second surface*, an environment that allows users to create augmented collaborative content mapped to real places with mobile devices.

The *ViiBoard* [23] is a remote collaboration tool with a simple setup, using only a large touch display and an RGBD camera that is mounted next to it. The 2D mode shows the shared content next to the captured remote user. Whenever the remote user modifies the shared content, the writing hand gets scaled on the display (cf. Fig. 2.4). In this way, occlusion of the

(a) Before transformation (b) After transformation

Figure 2.3: The scene before activation of the *IllumiRoom* system [11] (a). The *IllumiRoom* system increases the contrast of the scene through projecting color on the objects of the environment (b) .

content by the remote user is avoided while information about the current writing activities of the remote user is preserved.

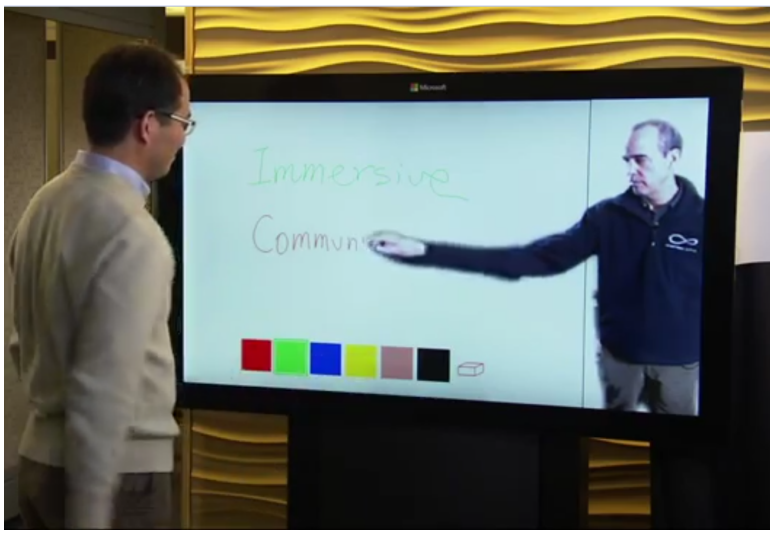


Figure 2.4: The *ViiBoard* transmits a simple RGB-image of the remote user [23]. On editing the content, the writing hand of the remote user gets scaled and follows the writing. This technique facilitates following the context and avoids interference of the collaborators.

ImmerseBoard [5] is the successor of the *ViiBoard*. It is a system for remote collaborate work based on a simple HW setup. All that is required is a Kinect v2 camera and a big touch display. The camera for the first participant is positioned on the right-hand-side of the screen; the camera of the second participant is positioned on the left-hand-side of the screen. This setup is required for a more realistic visualization of the remote participant.

The system allows the user to choose between four different modes. The *video mode* shows the remote participant in the side of the screen where the camera is positioned. So the participants almost have eye contact when they look into the camera. On the video the background is removed the remote participant is cropped and positioned. The *hybrid mode* is an enhancement of the video mode. The arm which is used for the interaction with the digital whiteboard is stretched if it is within a certain distance to the board. When the board is touched the hand points exactly to the touch point. This makes it easier to follow the remote collaborator when he is interacting with the whiteboard. The *tilt board mode* shows the full sized remote collaborator. This requires more space on the display compared to the "hybrid" mode and that's why the virtual board needs to be tilted. Apart from that the concept is the same as for the "hybrid" mode. The computation of the arm stretch has to consider the tilt of the board. The remote collaborator is more present but the content of the virtual board has a perspective distortion caused by the tilt. The *ImmerseBoard* allows better expressions through pointing, gaze direction and other gestures, and provides additional information through body posture, proxemics and eye contact. Furthermore, its setup integrates easily in existing offices.

The *mirror mode* simulates the real-world mirror. It seems as the remote collaborator stand next to the local collaborator and he can be seen trough the collaborative board which behaves mirror. The remote collaborator is transparent and blended over the content of the board. Because the participants look straight at the board, parts of the face cannot be recorded by the camera. The bodies themselves are reconstructed using the 3d point cloud from the Kinect camera. The quality could only be improved by using additional camera.

Valadares et. al. [18] analyze the IBM collaboration tool *Sametime 3D*. The tool allows a user to control a 3D avatar within a virtual world. Collaboration entirely takes place in the virtual environment, all users having avatars representing themselves. Documents, videos or flip charts can be shared in virtual meeting places (cf. Fig. 2.5). Additionally, the system supports spatial voice and highlights the currently talking user' avatar. Similar to real world scenarios, users can express their interest by spatial proximity to the speaker. The only relation to the real person is the avatars' voice. Natural gestures and movement such as walking, hand gestures and head movement can be achieved via mouse and keyboard inputs.

A vision-based system introduced by Izadi et al. [10] provides a combination of bi-manual and tangible interaction. It uses a tablet computer as a shared collaboration interface and a separate chat screen to display the remote user (cf. Fig. 2.6). A simple webcam captures the user while a top-down stereo camera captures the tablet surface. Virtual hands provide information about the remote users' actions and intentions. The system uses machine learning algorithms to recognize fingertips and touch interac-

Figure 2.5: Users represented by avatars move within the *Sametime 3D* virtual environment [18]. Additionally, the system supports shared contant, spatial voice and avatar gestures.

tion with the tablet surface. Additionally, physical objects such as pocket cameras are recognized and prompt a predefined action.

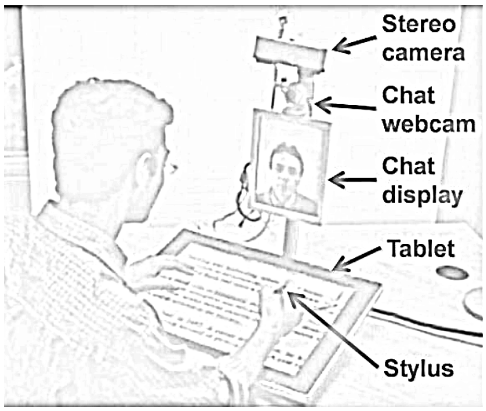


Figure 2.6: The *C-Slate* system setup [10]. The top-down stereo camera captures the tablet surface while the webcam captures the face. The tablet display shows shared and remote content, such as the hands of the remote user. Gaze awareness is provided through the chat display.

Zhang et al. [20] developed a teleconference system which imitates a face-to-face conversation. The system supports life-size 3D video, spatial audio and provides a virtual environment that covers furniture, lighting and wall colors. In this way, the illusion of sitting on the same table is created. The virtual conference room is rendered from each users' view-port

separately for correct motion parallax. Therefore, an eye-tracking algorithm is implemented (cf. Fig. 2.7). Spatial audio maps the voice of the currently talking user to its location within the virtual space.

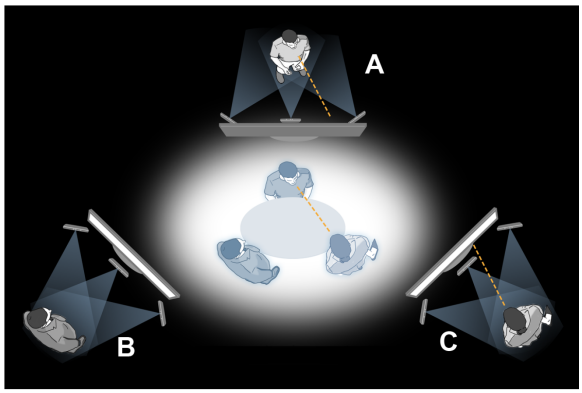


Figure 2.7: The figure shows *Viewport* by Microsoft Research [20]. Each participant A, B and C has a computer monitor as well as three cameras in front. Through eye-tracking, the system recognizes eye-contact between user A and C. Consequently, user B sees only a side view of user A and C.

2.2 Analysis of existing work

Recent work on collaborative systems differ in

- the remote collaboration ability,
- camera position (behind, on top, in front of, or next to the display),
- the environment (background removed or artificially created),
- proxemics (distance to the remote collaborator),
- content-user-relation (where is the user located in relation to the content),
- gaze awareness or eye-contact,
- gestures,
- number of users supported (one-to-one or multiuser),
- the dimension of the environment (2D or 3D), or
- setup and hardware requirements.

As previously stated, gestures and mimics can influence communication strongly. However, not all presented systems offer access to this meta-information. *Dynamo* has a simple and intuitive setup and presents inventive approaches about information sharing. It does not provide meta-information of the collaborators and does not support remote collaboration.

Likewise, *IllumiRoom* and *RoomAlive* also provide no functionality for remote collaboration. They offer unique techniques that could be used for remote collaboration, such as body tracking, object recognition and touch detection.

ConnectBoard and *ClearBoard* offer sound solutions for remote collaboration. Gestures and gaze awareness are provided. The systems require special hardware for setup and only support one-to-one collaboration. As RGB images are used, the quality of the user segmentation heavily depends on lighting conditions.

ViiBoard, its successor *ImmerseBoard* and *Viewport* segment the users using depth information respectively. They provide both gestures and mimics. Gestures in the *ViiBoard* system are co-linear to the produced content as the hand follows the writing in real-time. Touch screen and RGBD camera are the only hardware requirements. The communication is based on one-to-one collaboration. Proxemics are invalidated through the close distance to the screen, and therefore the remote user. *Viewport* is not concerned about shared content but supports multiuser interaction. The collaborators and the environment are represented in a virtual 3D world. This enables viewport dependent rendering of the environment. The system uses commonly available hardware components.

Sametime 3D imitates real-life conferences in a completely virtual environment. It supports 3D collaboration and gestures. However, the social presence in the environment is very abstract and communication does not feel natural to the users.

C-Slate needs multiple commonly available components for setup and offers only one-to-one communication. Mimics are supported whereas gestures are not available as meta-information. However, actions and intentions of the remote user can easily be retraced on the tablet surface. Additional intelligent computer vision algorithms provide various possibilities for collaboration.

Although the presented approaches offer interesting solutions to collaborative work, non of them provides a system that meets all criteria mentioned above. Most importantly, eye-contact, gestures and multiuser support for remote collaborative work.

Chapter 3

Methodology

A prototype to enhance remote collaboration was developed. This chapter describes the concept, the setup and the components of the prototype in detail.

3.1 Concept

Many video conferencing tools that support remote collaboration lack a linkage between the gestures of the participants and the content that is shared. For this reason, a prototype has been developed that allows the user a visually enhanced video conference experience by directly connecting the interacting hand to the produced content. In applications with multiple users, this approach allows the participants to quickly associate writings with the correct person. The perspective parallax is compensated for. In this way, the participants do not need to watch for moving mouse cursors and are able to quickly associate the writing with its owner.

A remote collaboration tool with enhanced user visualization was created. The system is targeted at conventional offices, adapting to the environmental conditions. The required components are

- a large collaboration screen (wall surface, touch display screen, or similar) that is able to detect touch input (such as the Microsoft Surface Hub [27]) or a projection screen with pen pairing (such as the CASIO® interactive whiteboard [22] or the w'inspire® systems [24]),
- an input device (such as bluetooth pen) or direct touch input (depending on the type of screen),
- a video conferencing tool (such as Skype®), and
- an RGBD sensor to capture the collaborator (such as the Microsoft® Kinect® v2 sensor).

The touch displays or projection screens with pen pairing deliver reliable information about the touch coordinates on the wall. The Kinect sensor

meets all desired requirements for the system, such as

- a given software development kit (SDK),
- a depth sensor to recognize the collaborator,
- fully developed hardware,
- maintained framework,
- global availability, and
- a manageable size.

The system needs to be available for all participating collaborators. Figure 3.1 shows an example for the setup of the system.

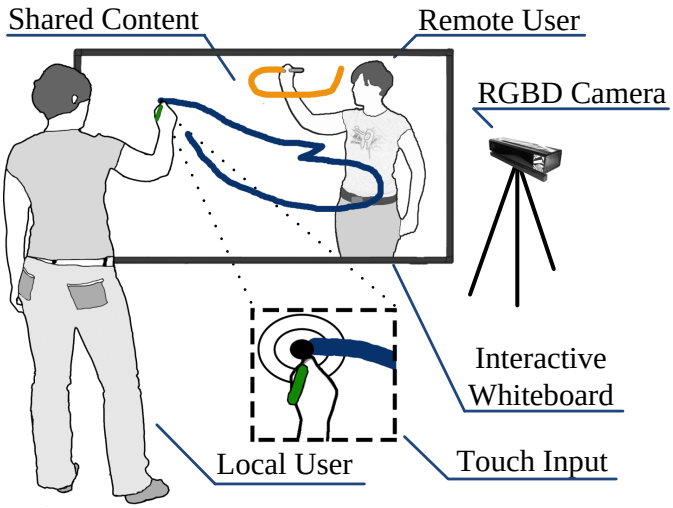


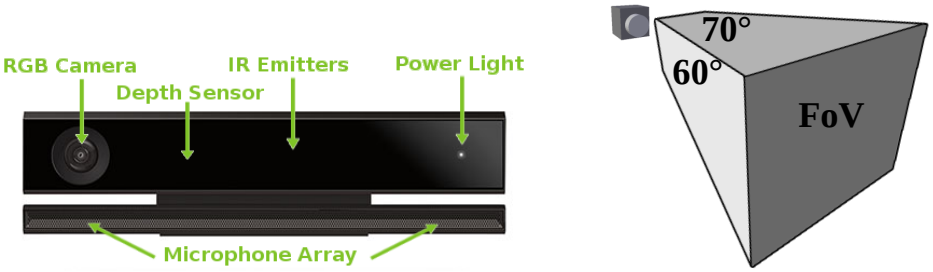
Figure 3.1: The remote collaboration scenario with the an RGBD camera placed next to the wall surface. The user and the remote conference participants work together on a collaborative surface.

3.2 Setup

It is assumed that the user participates in a video conference. All conference participants are connected to an application that allows them to write and draw together on one screen. Each user is captured by a sensor that is placed next to the display wall, facing the user from the right front, the side of the writing hand. The wall recognizes inputs via touch or bluetooth pen (cf. Fig. 3.1) and passes the touch input coordinates on to the application.

3.2.1 User caption with the MS Kinect v2 sensor

The Microsoft Kinect v2 sensor (cf. Fig. 3.2a) offers several image capturing hardware components as well as a SDK for the implementation of custom applications. The RGBD camera has a field of view (FoV) of 70° horizontally and 60° vertically (cf. Fig. 3.2b) [28]. All specifications and the functionality described in this section origin from the Microsoft Developer Network [25], [26].



(a) Kinect v2 sensor (b) Kinect FoV

Figure 3.2: The Microsoft Kinect for Windows v2 sensor (a) has a field of view of 70° horizontally and 60° vertically (b).

The sensor provides the following image streams (cf. Figure 3.3).

- Color source.
- Infrared source.
- Depth source.
- Body index source.
- Body source.
- Audio source.

Color source

The color source (cf. Fig. 3.3a) delivers a high-definition video stream with a resolution of 1920 × 1080 pixel. Based on the lighting condition, 30 or 15 frames per second (fps) are provided. The pixel values are able to store different color formats such as RGB, RGBA or YUV that are all saved as 8 bit unsigned integer per color channel.

Infrared source

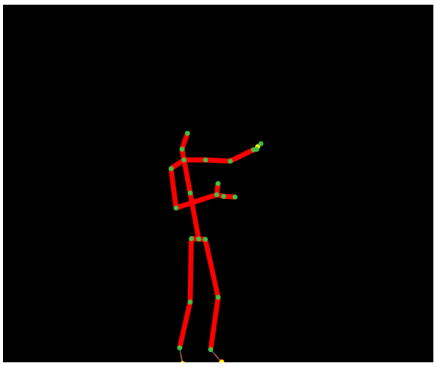
The infrared (IR) source (cf. Fig. 3.3b) provides a grayscale image with a resolution of 512 × 424 pixel at 30 fps. The pixel values store the IR intensity values as 16 bit unsigned integer. To illuminate the image consistently,

(a) Color source (16 : 9)

(b) Infrared source (4 : 3)

(c) Depth source (4 : 3)

(d) Body index source (4 : 3)



(e) Body source (4 : 3)

Figure 3.3: The sources provided by the Microsoft Kinect v2 sensor.

the ambient light has been removed. As the common light frequencies are removed, only the high frequencies pass. This guarantees the same base illumination for all lighting conditions. The procedure emphasizes the texture of the image and is therefore suited for facial recognition algorithms, reflec-

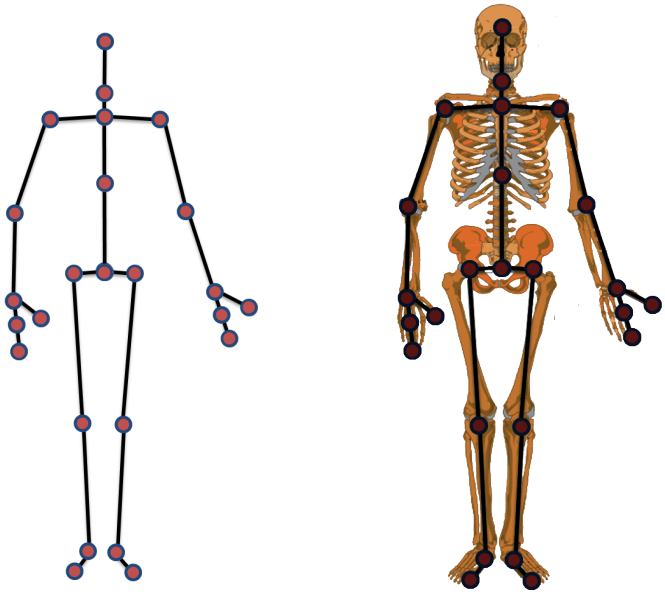
tive markers tracking and others. Depth frames and IR frames are perfectly aligned as the image data is derived from the same sensor and therefore has the same resolution. The sensor ranges from 0.5 meters to 8 meters, outside that scope no data is delivered.

Depth source

The depth source (cf. Fig. 3.3c) offers the same resolution (512×424) and range (0.5–8 meters) as the IR source, and a framerate of 30 fps. The pixel values are stored as 16 bit unsigned integer and represent the depth data, which is the distance in millimeters from the sensor's focal plane. The different shades of gray within the depth image indicate different depth values. All regions with the same gray level have the same distance from the sensor.

Body source

The body source (cf. Fig. 3.3e) reveals information about the skeleton of a body (cf. Fig. 3.4) at a framerate of 30 fps. One body consists of 25 known joints, each of them storing information about its position in the 3D space and its orientation. The scope ranges from 0.5 meters to 4.5 meters. The sensor can recognize up to six bodies simultaneously and the hand states (such as closed or open) of two bodies at the same time.



(a) Body joints (b) Skeleton with joints

Figure 3.4: The joints provided by the body source (source: [26])

Body index source

The body index source (cf. Fig. 3.3d) indicates a recognized body within the sensor's field of view and is computed based on the depth image. A recognized body's contour is filled with a unique index value to distinguish between body and background pixel. The maximum number of bodies to be recognized is six. The indices dedicated for the bodies range from zero to five, a value greater than five indicates a background pixel. The index is stored as 8 bit unsigned integer in the body index frame.

Audio source

The Kinect v2 sensor includes a microphone array with a steerable cone of focus for audio. However, the audio source has not been used for the prototype, as it focuses on enhanced gesture visualizations.

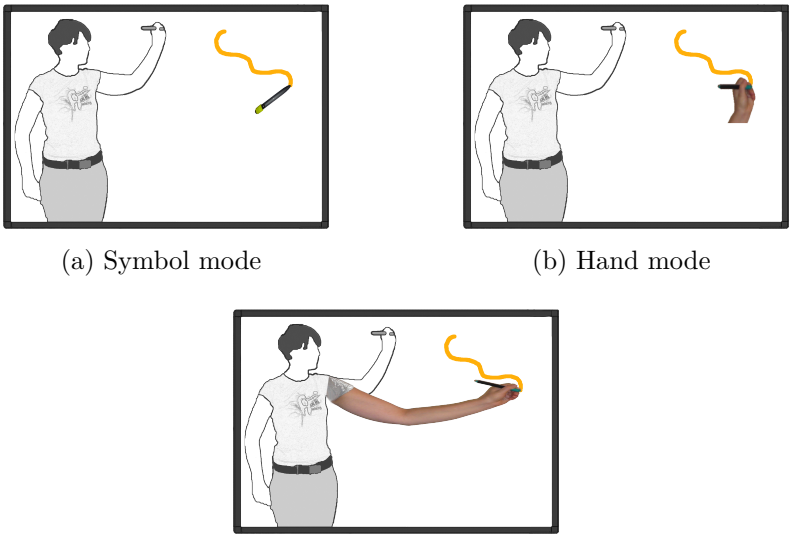
3.3 Remote visualization

In an ideal environment, the camera would be placed behind the screen to capture the user from the front, but state of the art wall surfaces are not lucent. With the camera positioned at the right front of the collaborative wall (cf. Fig. 3.1), the captured images shows a perspective parallax: All lines meet at one point at infinite depth. As a result, a point on the display screen from the camera's perspective does not match the same point on the display screen. That means when the display receives a touch input, the input coordinates do not match the coordinates from the video stream. In order to match the touch input coordinates with the finger tip coordinates from the sensor, the user's arm in the video frame has to be translated to the exact position of the touch input.

All modes accompany the users writing and replace the conventional mouse cursor with a video or an image. The symbol mode (cf. Fig. 3.5a) replaces the cursor with an image of choice and does not require any additional camera setup. The hand mode (cf. Fig. 3.5b) and the arm extension mode (cf. Fig. 3.5c) need a sensor to track and capture the user respectively. In this way, the captured image is manipulated by the application and shown to the remote user.

The manipulated frame is linked to the user and the user's gestures are visible at all time. In this way, it is easier for the collaborator to perceive at what point in time and on what region the remote user starts writing or drawing onto the collaboration surface.

The remote user is blended with the shared content and rendered onto the display. Additionally, the user can adjust the transparency of the remote users' representation in the GUI (cf. Fig. 3.6).



(a) Symbol mode (b) Hand mode

(c) Arm extension mode

Figure 3.5: The different modes to emphasize the cursor of the remote user. The symbol mode (a) replaces the cursor with an image of choice. The hand mode (b) duplicates the right hand of the user and copies it to the touch input position. The arm extension mode (c) provides a manipulated arm that is scaled and rotated to the exact touch input position.

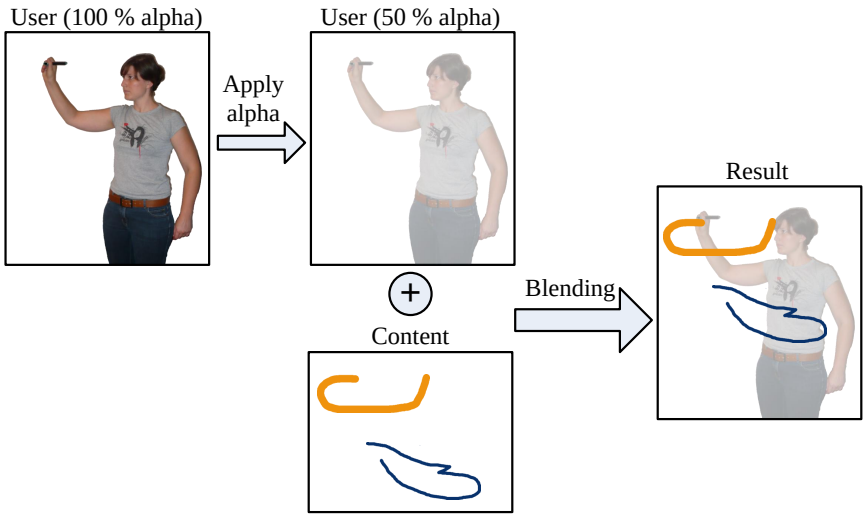


Figure 3.6: The opaqueness of the remote user is defined in the GUI and blended with the shared content.

Chapter 4

Prototype implementation

A prototype has been created to investigate the adaptability of transformed motion images in video conferencing software. A Microsoft Kinect v2 sensor was used to capture the image data. This chapter gives an overview of the computation pipeline and explains the implemented scenarios and used algorithms in detail. The user can choose between

- a symbol pointer that follows the input device (cf. Fig. 3.5a),
- a representation of the hand that is used as a cursor and follows the pointing device (cf. Fig. 3.5b), and
- a mode to scale and rotate the lower arm to the input point (cf. Fig. 3.5c).

For the prototype implementation, it is assumed that the user writes with the right hand. Different algorithms have been implemented to examine the performance on high-resolution images (1920×1080) as well as on low resolution (512×424) images. An image I is defined by

$$I = \{p = (x, y) \in \mathbb{N} \mid 1 \leq x \leq m, 1 \leq y \leq n\}, \tag{4.1}$$

with m and n stating the image width and height respectively, depending on the chosen image resolution.

The application aims to transform the video source in a way that the background is eliminated and the collaborator is extracted. The user's gestures accompany the writing on the collaborative surface. Therefore, the data passes different stages of computation (cf. Figure 4.1):

1. Acquisition of data
2. Transformation of the frame regarding the selected scenario
3. Rendering the transformed frame on the collaborative surface

These stages are explained in detail below.

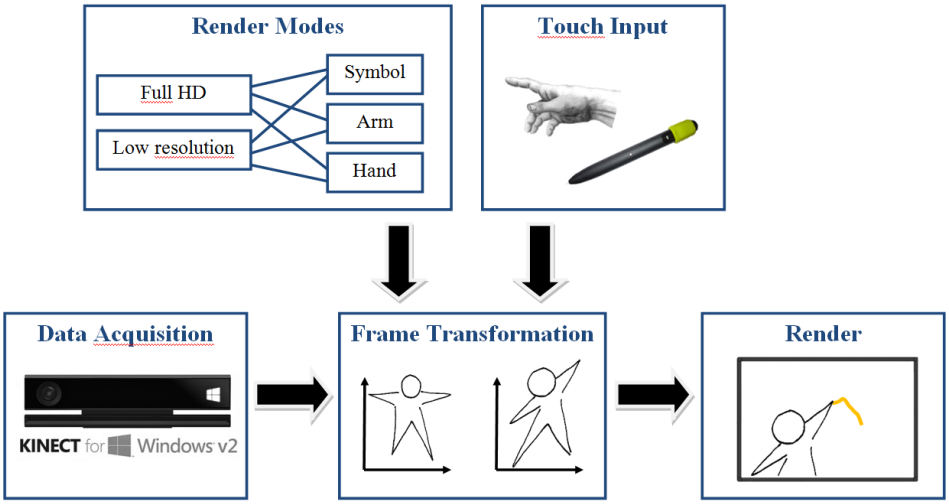


Figure 4.1: The image manipulation pipeline.

4.1 Data acquisition

A video stream that is acquired directly from the sensors shows the user and its entire environment within the camera's viewport (cf. Fig. 3.3a). To manipulate the image, all sources described in Chapter 3 are used for the computation, except for the audio source that is not required for the prototype implementation, and the IR source as the depth source that is derived from IR delivers all required information. The following sources are needed for further processing:

- **Color source.** All video manipulation scenarios defined above require a colored output, therefore the color source is needed.
- **Depth source.** Information about the silhouette of the user is stored in the body index source, which is useful for extracting the background from the user within the color image.
- **Body index source.** The color source and the body index source have different resolutions and origins – a coordinate mapper is required for mapping the points of one space into another. The mapper requires the depth source for operating.
- **Body source.** The body source is required for transforming particular parts of the body. The sensor offers access to so called *body joints* that indicate 25 different body parts (cf. Sec. 3.2.1).

The system requirements for using the Kinect v2 sensor are identified in Appendix A.

4.1.1 Determination of the camera position

In order to find the ideal camera position, several setups have been examined. Figure 4.2 shows the sensor positioned in various angles to the display in order to compare the different camera views of the user.

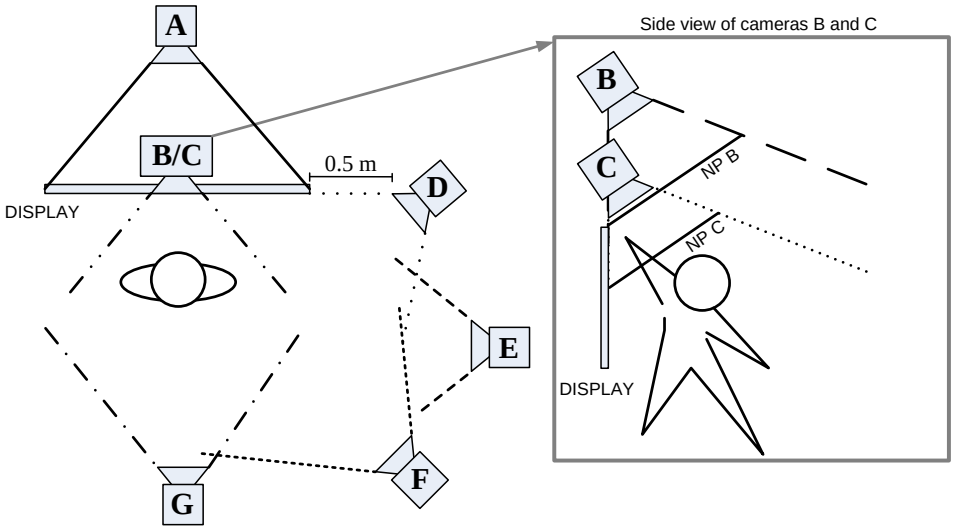


Figure 4.2: In order to capture the user from the ideal angle, different camera setups have been evaluated.

The camera positions are compared as follows:

- **Camera A** captures the user and the shared content respectively, assumed that the display is transparent. Drawbacks of this setup are the required space, the combined capture of user and content that needs to be further processed to be able to transform the user, and the display itself, as transparent interactive displays are not yet available globally.
- **Camera B** and **camera C** are positioned above the display in front of the user. Camera C, that is placed on the upper edge of the display, has a good view of the user and captures gestures and mimics respectively. However, the near plane of the camera has to be considered when the user interacts with the upper region of the display. In this case, all body parts that are closer to the camera than the near plane are discarded. In contrast, camera B does not interfere with the user. Due to the increased distance, the user's face and mimics are hardly visible and the view of the user is perspectively distorted.
- **Camera D** shows the user from front right. Most of the gestures and mimics are visible, and the skeleton is correctly detected. In addition, this position shows much of the environment. According to the shift

to the display, perspective distortions occur compared to a view from camera A (cf. Fig. 1.1b) or camera G.

- **Camera E** and **camera F** show a side and rear side view of the user. The sensor is not able to identify all limbs of the body correctly as they are occluded.
- **Camera G** shows the user from the rear, opposite to the display. It correctly detects the skeleton, but it looses track of the hands and arms when the user is writing on the display. Opposed to camera C and camera E, no perspective distortions occur. However, this setup does not allow the camera to capture the user's mimics.

Considering the advantages and disadvantages of the camera positions described above, the scenario of camera D is considered for the prototype implementation as it shows the most natural view of the user. In order to adjust the perspective distortion, algorithmic solutions are examined in the following section.

4.2 Frame transformation

This stage prepares the incoming data from the sensor for the remote transmission. The user can set the preferred parameters in the GUI.

4.2.1 Input parameters

The user can choose between two video qualities and three different representations of the virtual conference partner. Concerning the quality, the user can choose between

- full high-definition resolution (1920×1080), and
- low resolution (512×424).

One of the following three scenarios can be chosen to represent the remote partner:

1. **Symbol mode.** The point where the user touches the collaborative surface is marked with a symbol that follows the movements of the input.
2. **Hand translation mode.** Similar to the symbol mode, but instead of a symbol, the input movement is followed by a duplication of the users' own hand.
3. **Arm extension mode.** Similar to the the approach of Higuchi et al. [5], the lower arm of the user is extended to where the user draws on the collaborative surface.

4.2.2 Frame access

The frame sources can be addressed individually via a dedicated frame reader or all sources combined via a multi source frame reader. Individually addressed frames need to be synchronized manually while the multi source frame reader synchronizes the frames automatically and allows the application access to all chosen frames on a single event. Each frame has a descriptor which stores metadata information such as the width and the height of the image stream. The content of the frames is stored in 1-dimensional buffers. Therefore, rows and columns of the images have to be calculated manually from the index.

The coordinate mapper, provided by the Kinect SDK, is required for mapping the points of one coordinate system to another. The points of the different coordinate systems are

- color space points (originate from color source),
- depth space points (originate from depth, IR or body index source), and
- camera space points (originate from body source).

The conversion from one coordinate system to another is fundamental, as body joints are provided in depth space and the color frames are provided in color space. To find out what color pixel belongs to a body joint, the coordinate mapper delivers the respective pixel to look up.

As the coordinate systems provide frames with different resolutions, the coordinate mapper contains a look-up table to find the equivalent pixel positions of one coordinate systems within another. A manual calibration of the sources is redundant, as the mapper stores information about the sensor's intrinsic and extrinsic parameters.

Not every pixel of one coordinate system has an equivalent within another. The RGB camera delivers wide angle picture streams in a ratio of 16:9, therefore the other streams that have a ration of 4:3 do not have corresponding pixel at the left and the right boundary of the color image. Figure 4.3 demonstrates the matching and non-matching regions between the color space stream and the streams containing depth space points (depth, IR and body index source). Several boundary pixel cannot be matched and thus store an infinity value. Therefore it is important to check if the pixel have valid values when traversing the image.

In order to get the corresponding depth space pixel positions to a color frame, the mapper is used by

```
1 //depth data stream
2 private ushort[] depthDataSource;
3
4 //number of pixel in HD color stream
5 int colorImageLength = 1920 * 1080;
6
```

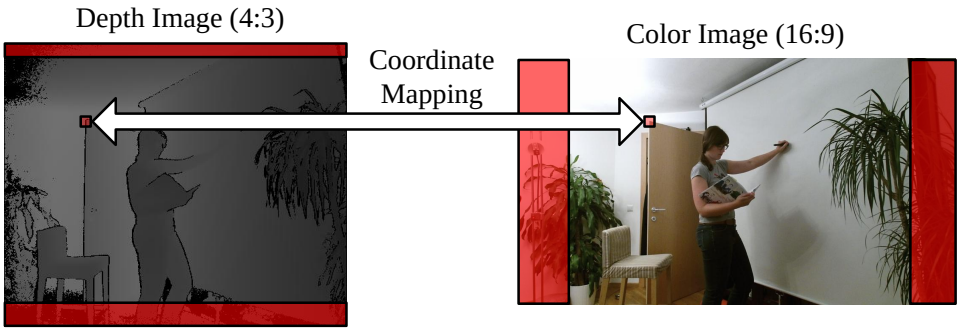


Figure 4.3: The depth space points do not have an equivalent within the color space source for each pixel and vice versa. The colored area shows the boundary regions that cannot be matched.

```
7  //coordinate mapper for mapping color space points to depth space points
8  private DepthSpacePoint[] colorToDepthSpaceMapper;
9  colorToDepthSpaceMapper = new DepthSpacePoint[colorImageLength];
10
11 //the colorToDepthSpaceMapper-array is filled up with look-up position values
12 coordinateMapper.MapColorFrameToDepthSpace(depthDataSource,
       colorToDepthSpaceMapper);
```

Subsequently, when traversing the color frame, the corresponding pixel positions for depth, IR or body index can be obtained as following:

```
1 int colorSensorBufferWidth = 1920;
2 //determine the index position of the color buffer
3 int idxCurrColorPixel = yCurrent * colorSensorBufferWidth + xCurrent;
4
5 //obtain depth pixel via coordinate mapper
6 float xDepthPixel = colorToDepthSpaceMapper[idxCurrColorPixel].X;
7 float yDepthPixel = colorToDepthSpaceMapper[idxCurrColorPixel].Y;
```

The corresponding color space pixel positions to a depth frame can be obtained similarly using

```
1  //depth data stream
2  private ushort[] colorDataSource;
3
4  //number of pixel in HD color stream
5  int depthImageLength = 512 * 424;
6
7  //coordinate mapper for mapping depth space points to color space points
8  private ColorSpacePoint[] depthToColorSpaceMapper;
9  depthToColorSpaceMapper = new ColorSpacePoint[depthImageLength];
10
11 //the depthToColorSpaceMapper-array is filled up with look-up position values
12 coordinateMapper.MapDepthFrameToColorSpace(depthDataSource,
       depthToColorSpaceMapper);
```

4.2.3 Background elimination

The user's body is detected to eliminate unnecessary information such as the background, and to ease the selection of user body parts for manipulation. In order to extract the body from the background, the information in the body index frame (cf. Fig. 3.3d) is used. The color of each pixel p that belongs to a body b is written directly into the back buffer. A coordinate mapper (cf. Sec. 4.2.2) is used to check if p has a corresponding body index that indicates a body. As the information about the body is stored in the body index space as depth space points, the pixel position in color space has to be mapped to the depth space. The resulting image shows a person in RGBA color while the background is discarded.

The sensor's SDK offers an interface to read the video stream frame by frame. Each valid frame is manipulated according to the mode chosen (cf. Figure 4.6). Independent of the chosen scenario, a background elimination is performed to extract the user from the environment. Subsequently, the sensor detects the body, as the body index source does not contain color information, the body index frame and the color frame have to be mapped to each other. The coordinate mapper contains information about what pixel of the body index belongs to what pixel in the color source (cf. Chapter 4.2.2). By matching the pixel of the body index with the color source pixel, all color pixel that do not match a body index are discarded. Consequently, the target image shows the user in color without background (cf. Figure 4.4).

(a) With background (b) Background eliminated

Figure 4.4: The color image from the Kinect sensor before (a) and after background elimination (b).

Figure 4.5 shows the look-up procedure that has to be performed for each pixel. The color target buffer is traversed sequentially pixel by pixel. For each pixel, the following steps have to be performed:

1. **Body index look-up.** It is checked if the current pixel is part of a body. If this pixel does not belong to a body it is discarded.
2. **Depth space to color space mapping.** The color of the pixel is looked-up in the color frame using the coordinate mapper. If the

mapped pixel is invalid, it is discarded.

3. **Write back color.** The color of the pixel in the color source buffer is written in the corresponding pixel of the target color buffer.

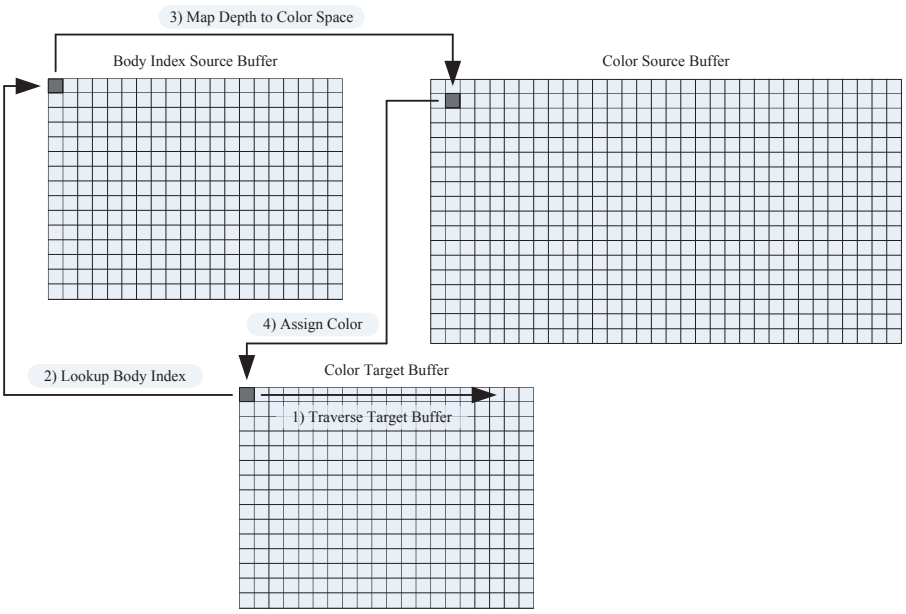


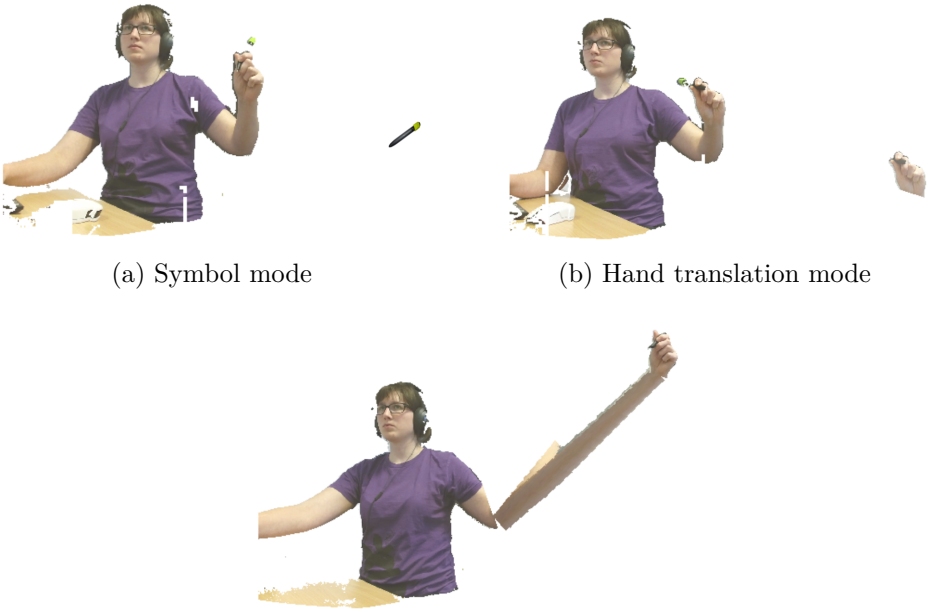
Figure 4.5: The three buffers that are required for the image manipulation process.

Subsequently, the colored body is transformed according to the chosen transformation mode (cf. Fig. 4.6). The wrist body joint is required to determine the position of the wrist, as well as the elbow joint that is required to determine the vector of the lower arm. The arm extension mode (cf. Fig. 4.6c) scales the lower arm to the input point. Therefore, the body is rendered without manipulation up to the right elbow.

In the prototype, the algorithm used to traverse the body index buffer can be chosen in the GUI. The different algorithms are compared and described in detail below. The performance of the presented algorithm depends on the number of pixel visited and on the number of checks if the pixel belongs to a body.

4.2.4 Scenario 1: Symbol pointer

The first scenario the user can choose is the symbol mode (cf. Fig. 4.6a). An adequate symbol is chosen in advance. The position where the symbol is rendered is updated simultaneously with the touch input coordinates on



(a) Symbol mode

(b) Hand translation mode

(c) Arm extension mode

Figure 4.6: One of the following modes replaces the mouse cursor: An arbitrary symbol can be chosen, eg. a pencil (a). The writing hand is duplicated and translated (b), or the writing arm is scaled to the input device's position (c).

each render cycle of the frame. Consequently, the symbol image follows the touch input movement. The body of the user is visible next to it.

4.2.5 Scenario 2: Hand duplication

As a second option, the user can choose the hand scenario (cf. Fig. 4.6b). In this mode, the user's hand is duplicated to the position of the touch point coordinates. The user's body is rendered next to it. Given the lower arm vector

$$\mathbf{a} = \begin{pmatrix} x_{wrist} - x_{elbow} \\ y_{wrist} - y_{elbow} \end{pmatrix} \tag{4.2}$$

from the elbow to the wrist, and the vector of current pixel $p_{current}$ to p_{wrist}

$$\mathbf{c} = \begin{pmatrix} x_{current} - x_{wrist} \\ y_{current} - y_{wrist} \end{pmatrix}, \tag{4.3}$$

the dot product between the two vectors $\mathbf{a} \cdot \mathbf{c}$ indicates if $p_{current}$ is above or below the normal of $\mathbf{a}$ (cf. Fig. 4.7). The algorithm is explained in more detail in Section 4.2.7.

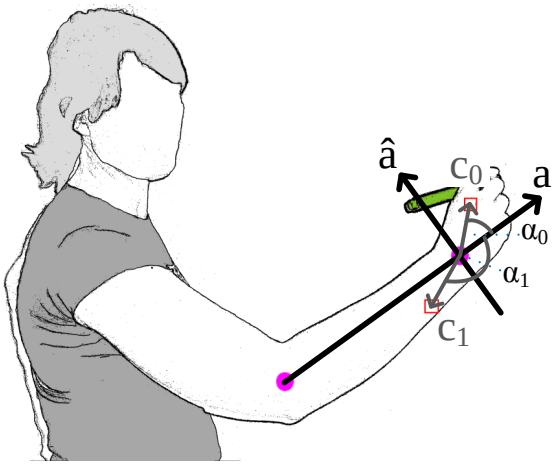


Figure 4.7: The dot product $\mathbf{a} \cdot \mathbf{c}$ indicates if the current pixel belongs to the hand or not.

In order to determine the distance the hand has to be moved, the translation vector $\mathbf{t}$ is calculated by

$$\mathbf{t} = \mathbf{p}_{touch} - \mathbf{p}_{handTip}. \tag{4.4}$$

Subsequently, all color pixel that belong to the hand are looked up in C and translated by $\mathbf{t}$.

4.2.6 Scenario 3: Arm transformation

The third scenario the user can choose is an arm extension mode (cf. 4.6c). If the user is not writing, the video recording of the body is transmitted without manipulations. If the remote user is interacting with the screen, the hand scales to the touch input point. The advantage of this scenario is that the association between hand and user is given at all times.

Due to the perspective distortion, the hand of the user is not in alignment with the touch input. This is corrected by scaling and rotating the arm to the touch input coordinates. In the prototype, two different approaches for the arm manipulation have been implemented:

- a pixel based approach, and
- a vector based approach.

These approaches are discussed below.

Pixel based arm transformation

In order to point to the touch coordinates, the arm of the user is first scaled, then rotated.

1. Scale First, the arm is scaled to the proper length in order to reach the touch point (cf. Fig. 4.9). The rotation is handled in a next step. The algorithm is scaling a sub-region of the color stream to the target buffer. The example describes the scaling of the right lower arm A. As a precondition it is assumed that all pixel on the right side of the elbow joint

$$\forall p \notin A \,|\, p_{x_i} < p_{x_{elbow}} \tag{4.5}$$

are scaled. When scaling the arm it has to be considered that the prolonged arm has to have the same length at all time. Using the same scale factor for all angles of the arm would result in an unproportional scale in one of the directions.

First, a scaling factor has to be determined for the x and y direction respectively. Therefore, the slope is taken into account. To get the slope parameters, the angle α between a horizontal line $\mathbf{h}$ starting from p_{elbow} and the arm vector $\mathbf{a}$ is calculated using the dot product [6]

$$\mathbf{a} \cdot \mathbf{h} = |\mathbf{a}||\mathbf{h}| \cos(\alpha) \Rightarrow \alpha = \arccos\left(\frac{\mathbf{a} \cdot \mathbf{h}}{|\mathbf{a}||\mathbf{h}|}\right) \tag{4.6}$$

The angle is used for the calculation of the scale factors

$$\text{factor}_x = s * \cos\alpha + 1 \tag{4.7}$$

and

$$\text{factor}_y = s * \sin\alpha + 1, \tag{4.8}$$

s indicating a scalar scaling factor that is dependent on the distance to the touch point. A lookup position and color for scaled regions is calculated subsequently. The distance of the current pixel to the elbow calculated by

$$\Delta_x = x_i - x_{elbow} \tag{4.9}$$

and

$$\Delta_y = y_i - y_{elbow}. \tag{4.10}$$

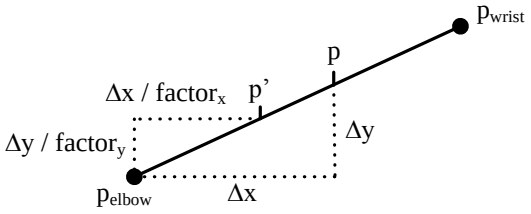


Figure 4.8: The scale factor is calculated using the sine for step size in the y-direction ($factor_y$) and the cosine for the step size in the x-direction ($factor_x$).

This offset is used to determine the lookup position (cf. Fig. 4.8)

$$x_i' = x_{elbow} + \frac{\Delta_x}{\text{factor}_x} \tag{4.11}$$

and

$$y_i' = y_{elbow} + \frac{\Delta_y}{\text{factor}_y}. \tag{4.12}$$

x_i' and y_i' are only used for the color look-up if the original pixel belongs to the arm. Otherwise, a standard color look-up with x and y is performed. Consequently, the arm is scaled to an appropriate length to reach the touch input. Subsequently, a rotation of the scaled arm has to be performed to exactly match the touch point.

(a) No manipulation (b) Scaled arm

Figure 4.9: (a) The body before transformation and (b) after the scale operation has been applied to the lower arm.

2. Rotation In the resulting target image, the arm has to point exactly to the touchpoint. Therefore, after the arm has been scaled, it has to be rotated to the touch coordinates additionally. The pivot point of the rotation is the elbow joint p_{elbow} to ensure that the lower arm is connected to the upper arm. Figure 4.10 shows the images before and after the rotation.

Hearn and Baker [4] define a 2D-rotation matrix for the angle θ by

$$R = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta. \end{pmatrix} \tag{4.13}$$

In order to rotate the right arm A that belongs to the body of the remote collaborator, each pixel

$$p_i \in A, \quad i = 0, 1, \ldots, n, \quad A \subset I \tag{4.14}$$

has to be rotated. The rotation angle θ is determined by the angle between the arm vector $\mathbf{a}$ and the rotated target arm vector

$$\mathbf{a}_{rotated} = \begin{pmatrix} x_{touch} - x_{elbow} \\ y_{touch} - y_{elbow} \end{pmatrix}, \quad a_{rotated_0}, \ldots, a_{rotated_n} \in I \tag{4.15}$$

(a) No manipulation (b) Rotated arm

Figure 4.10: The image before (a) and after (b) the applied rotation of the right lower arm.

that is identified by the touch point of the input device (cf. Fig. 3.1). Based on these vectors, the angle from $\mathbf{a}$ to $\mathbf{a}_{rotated}$ is calculated by the dot product [6]

$$\mathbf{a} \cdot \mathbf{a}_{rotated} = |\mathbf{a}||\mathbf{a}_{rotated}| \cos(\theta) \Rightarrow \theta = \arccos\left(\frac{\mathbf{a} \cdot \mathbf{a}_{rotated}}{|\mathbf{a}||\mathbf{a}_{rotated}|}\right), \tag{4.16}$$

with $|\mathbf{a}|$ being the magnitude of $\mathbf{a}$. Subsequently, the rotation R is applied to $\mathbf{a}$ by

$$\mathbf{a}R(\theta) = \mathbf{a}_{rotated}. \tag{4.17}$$

The arm is correctly transformed within the resulting image but a regular pattern of white noise occurs (cf. Fig. 4.10b). This noise is caused by the inevitable translation of floating point positions resulting from the rotation R to fixed point positions that are required to draw the final pixel to the image.

To summarize, the pixel based arm transformation has the disadvantage that floating point operations cause noise due to rounding errors.

Vector based arm transformation

To overcome the errors of the pixel based arm transformation, a vector based arm transformation has been examined. The advantages are the following:

- Rotation and scale are combined in one operation.
- The region checks for each pixel are omitted. Instead, only the information of the start and end point of the real arm and the start and end point of the target arm are relevant for the calculations.

In order to calculate the virtually rotated and scaled target arm $\mathbf{a}_{target}$ out of the real arm $\mathbf{a}$ that is visible in the color stream, the vectors have to be

determined respectively. The vector for $\mathbf{a}$ is determined by

$$\mathbf{a} = \begin{pmatrix} x_{wrist} - x_{elbow} \\ y_{wrist} - y_{elbow} \end{pmatrix}, \tag{4.18}$$

the vector for $\mathbf{a}_{target}$ is determined by

$$\mathbf{a}_{target} = \begin{pmatrix} x_{touch} - x_{elbow} \\ y_{touch} - y_{elbow} \end{pmatrix}. \tag{4.19}$$

Figure 4.11 shows the real arm as it is retrieved from the color source and the target vector where the real arm is transferred to.

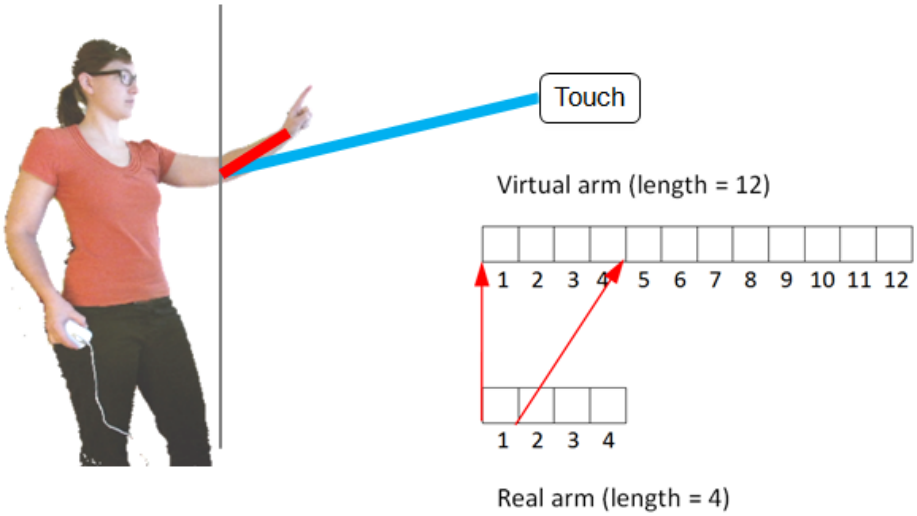


Figure 4.11: The values of the original vector $\mathbf{a}$ (red line) are subsequently applied to the virtual vector $\mathbf{a}_{target}$ (blue line).

The vector based algorithm consists of the following steps:

- Determination of the vectors $\mathbf{a}$ and $\mathbf{a}_{target}$.
- Reading color from $\mathbf{a}$ and writing it to $\mathbf{a}_{target}$. Therefore, the vectors are traversed simultaneously. Depending on the scale factor, the step width for $\mathbf{a}$ is increased (target arm shorter) or decreased (target arm longer).
- For each pixel on $\mathbf{a}_{target}$, the normals are used to fill up the rest of the arm.
- Definition of appropriate boundaries. The start and the end point of the transformation are set.

The adequate step size or sampling size that is used for stepping along $\mathbf{a}$ is determined by the magnitude of $\mathbf{a}$ and $\mathbf{a}_{target}$ – the longer vector determines the step width by

```
1 float totalSteps;
2 if (vArmLength < vTargetArmLength)
3     totalSteps = (float)(vTargetArmLength);
4 else
5     totalSteps = (float)(vArmLength);
6 float stepSizeArm = vArmLength / totalSteps;
7 float stepSizeTargetArm = vTargetArmLength / totalSteps;
```

To guarantee a noise free result, the total step size can be oversampled by multiplying `totalSteps` with the factor 2 according to the Nyquist sampling interval described by Hearn and Baker [4].

The normals of $\mathbf{a}$ are defined by the left normal

$$\hat{\mathbf{a}}_{left} = \begin{pmatrix} y \\ -x \end{pmatrix}, \tag{4.20}$$

and the right normal

$$\hat{\mathbf{a}}_{right} = \begin{pmatrix} -y \\ x \end{pmatrix}, \tag{4.21}$$

considering that the origin of the coordinate system is the upper left corner of the image. Figure 4.12 shows the normals $\hat{\mathbf{a}}_{left}$ and $\hat{\mathbf{a}}_{right}$ result of the applied algorithm depictively.

Figure 4.12: When the target arm vector is fully traversed, the arm is stretched and rotated, and points to the touch point. The original arm $\mathbf{a}$ is indicated by the short red line, the target arm $\mathbf{a}_{target}$ is indicated by the long blue line.

In case the normal $\hat{\mathbf{a}}_{left}$ points into the body, a suitable boundary has to be set as explained in detail below (cf. Fig. 4.13). Compared to the pixel based arm transformation, the floating point errors do not occur, as each pixel of $\mathbf{a}_{target}$ gets a corresponding color value assigned.

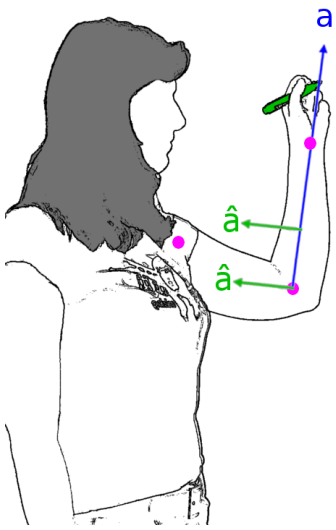


Figure 4.13: In the region of the elbow, the normal vector points into the body. Without an appropriate termination criteria, parts of the upper arm are transformed along with the lower arm.

4.2.7 Determination of accurate boundary vectors

In case the normal $\hat{\mathbf{a}}_{left}$ points into the body (cf. Fig. 4.13), a suitable boundary has to be set. Furthermore, the region of the hand has to be separated from the region of the lower arm in order to prevent the hand from the transformations that are applied to the lower arm. Two boundary vectors have to be evaluated:

1. The lower boundary located at the elbow (cf. Fig. 4.14a), and
2. the upper boundary located at the wrist (cf. Fig. 4.14b).

The lower boundary $\mathbf{b}_{lower}$ is most likely to interfere with the body as $\hat{\mathbf{a}}_{left}$ points into the upper arm in the region of the elbow point. The standard algorithm steps along $\hat{\mathbf{a}}_{left}$ until it reaches a background pixel. In this case, without a limiting boundary for $\mathbf{b}_{lower}$, the algorithm continues reading pixel of the upper arm. Those pixel would also enter the next step of transformation and would get scaled and rotated. In the rendered image artefacts would occur. An improved termination criteria is the half vector between the shoulder vector

$$\mathbf{s} = \begin{pmatrix} x_{shoulder} - x_{elbow} \\ y_{shoulder} - y_{elbow} \end{pmatrix} \tag{4.22}$$

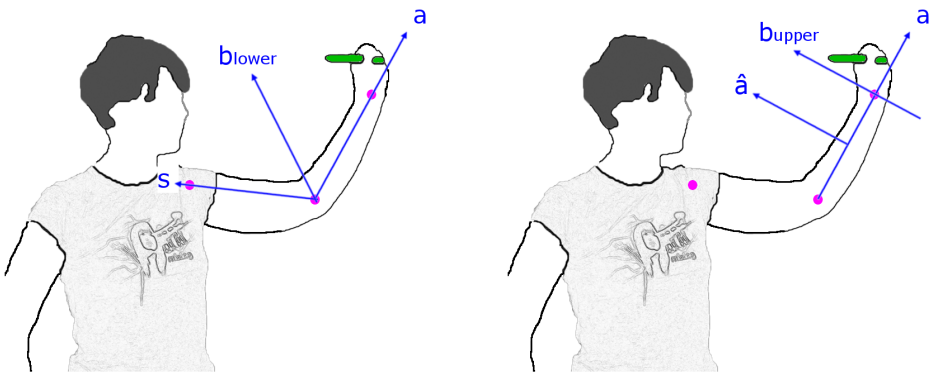
and the lower arm vector $\mathbf{a}$. It is calculated as follows:

$$\mathbf{b}_{lower} = \frac{\mathbf{s} + \mathbf{a}}{|\mathbf{s} + \mathbf{a}|}. \tag{4.23}$$

The upper boundary $\mathbf{b}_{upper}$ is set to the vector normal

$$\mathbf{b}_{upper} = \hat{\mathbf{a}}_{left}. \tag{4.24}$$

As $\mathbf{b}_{upper}$ does not interfere with other body parts, the boundary is naturally limited by the wrist joint and does not have to be checked explicitly. However, the boundary criteria would not hold if the wrist touches other parts of the body or if it is placed in front of the body. A possible solution to this problem is presented in Section 4.2.8.

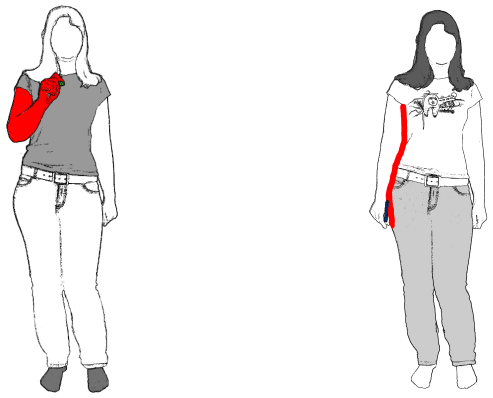


(a) Lower boundary (b) Upper boundary

Figure 4.14: (a) The lower boundary is defined by the half vector $\mathbf{b}_{lower}$ of the shoulder vector $\mathbf{s}$ and the arm vector $\mathbf{a}$. (b) The upper boundary is defined by the normals of the arm vector $\hat{\mathbf{a}}_{left}$ and $\hat{\mathbf{a}}_{right}$.

4.2.8 Overlapping body parts

When writing on the interactive whiteboard, the writing hand often occurs to be in front of the body (cf. Fig. 4.15a). This occurs to be a problem in case a flood fill algorithm is used to determine the body regions within the hand extraction mode as well as the arm extension mode respectively, as both use the background pixel as a termination criteria. When the hand is in front of the body, the hand is not sourrounded by background pixel. The only criteria that differs is the depth. a to determine the hand section. Furthermore, the termination criteria within the arm extension mode are no longer valid if the user places the arm parallel to the body, closing the gap between arm and body (cf. Fig. 4.15b). Consequently, the algorithm would not terminate as no background pixel is visited. The solution is to extend the abort criterion by the depth value. The sensor delivers the depth value in millimeters from its focal plane (cf. Sec. 3.2.1) and that value is used in combination with a predefined threshold t in order to determine if the point



(a) (b)

Figure 4.15: Areas colored in red show the conflicting regions that require an adapted algorithm. In order to be able to differentiate between hand and body region, the adapted algorithm uses the depth value (a). In case the depth values of hand and body are equal and no background pixel occurs in between them, the algorithm cannot distinguish between hand and body region (b).

belongs to the writing hand **a** or the body b. The difference of the depth values

$$\Delta d = I(p_i) - I(p_{i+1}) \tag{4.25}$$

is compared with the threshold and if the absolute value

$$||\Delta d|| > t, \tag{4.26}$$

the algorithm terminates. In this way an overflow of the hand fill is prevented, and the arm is extracted, scaled, and positioned correctly. However, if the hand has a contact to the body, the depth values of the arm cannot be distinguished from the rest of the body.

4.3 Rendering

The transformed image of the user is projected onto the collaborative surface. To guarantee a minimum occlusion of the collaborative content, each client is able to adapt the opacity of the remote user with a slider in the GUI. Displaying the remote user on the collaborative surface creates the effect of facing each other.

4.4 Performance enhancement

As high definition images are used within the prototype, the algorithm performs background checks and coordinate mappings for 1920×1080 pixel in the worst case. In order to increase the performance of the application, several considerations have to be taken, such as

- body detection is executed faster when using flood fill algorithms (cf. Sec. 4.4.1),
- recursive flood fill operations are started in their own thread to avoid a stack overflow,
- unmanaged code is preferred over managed code (cf. Sec. 4.4.2),
- color values are written directly into the back buffer,
- operations within a loop are to be kept to a minimum,
- primitive types are preferred over structures, and
- operations such as modulo and division are to be kept to a minimum.

4.4.1 Improved body detection using flood fill algorithms

In contrast to the sequential approach described in Section 4.2.3, flood fill algorithms do not have to visit each pixel of the body index buffer. Instead, they start at a given seed point s that lies inside the body, visit all neighboring pixel of s and terminate if a background pixel is hit. A seed point takes the value of a valid body joint. In case no body is detected, the sequential approach is executed and shows the unaltered color stream. Several flood fill algorithms have been examined that differ in the order the neighboring pixel are visited. The search patterns are

- recursive (cf. Alg. 4.1),
- breadth-first search (BFS, cf. Alg. 4.2), and
- depth-first search (DFS, cf. Alg. 4.3).

Recursive flood fill algorithm

The recursive flood fill algorithm (cf. Alg. 4.1) traverses the region by using the calculated parameters in a new call of the function itself [1]. A drawback of this algorithm is that the number of elements pushed onto the stack memory grows with each recursion, which could cause a stack overflow.

Breadth-first search

The iterative BFS algorithm (cf. Alg. 4.2) uses a queue to store the pixel to visit. Compared to the recursive approach, the risk of a memory overflow is exiguous as the object is stored in the heap memory. The pixel enqueued first are visited first (first-in first-out principle). The performance is slower

Algorithm 4.1: The recursive flood fill algorithm.

```
I =bodyIndexBuffer and backgroundPixel = 255
FloodFill(x, y)
    if coordinate (x, y) is within image boundaries
         and I(x, y) < maxNumberOfBodies then
        GetColorAndDrawIntoBackbuffer(x, y)
        FloodFill(x + 1, y)
        FloodFill(x − 1, y)
        FloodFill(x, y + 1)
        FloodFill(x, y − 1)
    else
        return
    end if
end
```

as many elements are queued until a termination criteria comes true [1, 14].

Algorithm 4.2: The BFS flood fill algorithm using a queue.

```
I =bodyIndexBuffer and backgroundPixel = 255
FloodFill(x, y)
    Create an empty queue Q
    Insert the seed point into Q: ENQUEUE(Q, (x)), ENQUEUE(Q, (y))
    while Q is not empty do
        Get next point p_next by dequeuing the first
         coordinate of Q: x ← DEQUEUE(Q), y ← DEQUEUE(Q)
        if p_next is within image boundaries
         and I(x, y) != backgroundPixel then
            GetColorAndDrawIntoBackbuffer(x, y)
            SetPixelVisited(p_next)
            ENQUEUE(Q, x + 1)
            ENQUEUE(Q, y)s
            ENQUEUE(Q, x − 1)
            ENQUEUE(Q, y)
            ENQUEUE(Q, x)
            ENQUEUE(Q, y + 1)
            ENQUEUE(Q, x)
            ENQUEUE(Q, y − 1)
        end if
    end while
end
```

Depth-first search

The iterative DFS algorithm (cf. Alg. 4.3) uses a stack to store the pixel to visit. Like the BFS algorithm, the object is stored in the heap memory. The pixel on top of the stack are visited first (last-in first-out principle). Therefore, all pixel are successively processed in one direction and therefore reaches the termination criteria at an earlier point in time than the BFS [1, 14].

Algorithm 4.3: The DFS flood fill algorithm using a stack.

```
I =bodyIndexBuffer and backgroundPixel = 255
FloodFill(x, y)
    Create an empty stack S
    Push the seed point onto S: PUSH(S, x), PUSH(S, y)
    while S is not empty do do
        Get next point p_next by pulling the first
         coordinate of S: x ← POP(S), y ← POP(S)
        if p_next is within image boundaries
         and I(x, y) != backgroundPixel then
            GetColorAndDrawIntoBackbuffer(x, y)
            SetPixelVisited(p_next)
            PUSH(S, x + 1)
            PUSH(S, y)
            PUSH(S, x − 1)
            PUSH(S, y)
            PUSH(S, x)
            PUSH(S, y + 1)
            PUSH(S, x)
            PUSH(S, y − 1)
        end if
    end while
end
```

Benchmark

The sequential, recursive flood fill, BFS and DFS algorithms have been benchmarked to find the best algorithm in terms of performance. The measurement criteria is the arithmetic mean of frames per second based on the frame duration fd

$$\overline{fps} = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{fd_i}, \tag{4.27}$$

n stating the number of samples taken, and the standard deviation from the mean [19]

$$s_n = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(fps_i - \overline{fps}\right)^2}. \tag{4.28}$$

Note that the sensor cannot process more than 30 fps with good lighting conditions (cf. Sec. 3.1). The performance of the different flood fill algorithms for a high resolution target buffer (1920×1080) is shown in Table 4.1, and for a low resolution target buffer (512×424) is shown in Table 4.2.

Algorithm	sequential	recursive	DFS	BFS
Frame duration (s)	0.09	0.05	0.06	93.46
$\overline{fps}$	11.10	20.88	18.05	0.01
s_N	1.20	3.57	1.29	0.01

Table 4.1: The performance of the different flood fill algorithms when rendering into a low resolution target buffer (1920×1080). The best performance is achieved by the recursive flood fill, followed by the DFS and the sequential algorithm. The BFS cannot process the large amount of data with reasonable speed.

Algorithm	**sequential**	**recursive**	**DFS**	**BFS**
Frame duration (s)	0.03	0.03	0.03	4.91
$\overline{fps}$	30.04	30.09	30.12	0.20
s_N	0.40	2.80	2.36	0.04

Table 4.2: The analysis of the different flood fill algorithms when rendering into a low resolution target buffer (512×424). The results show that the sequential, recursive and DFS algorithm are almost equal regarding their frame rate compared to the BFS that is comparatively slow.

A benchmark indicates that the recursive algorithm performs best when rendering into a high-definition target buffer T_{HD} and into a low resolution target buffer T_{LR} respectively. However, in high definition mode sufficient stack memory is required. The standard stack size of .NET applications is 1 MByte. The performance test with high resolution buffers required about 20 - 40 MByte of stack size, depending on the distance of the the body. A maximum stack size of 30 MByte has proven to be sufficient for the tests. A severe drawback is the risk of a stack overflow if the body area gets too big. This occurs if the user approaches the sensor and thus, the body area gets bigger.

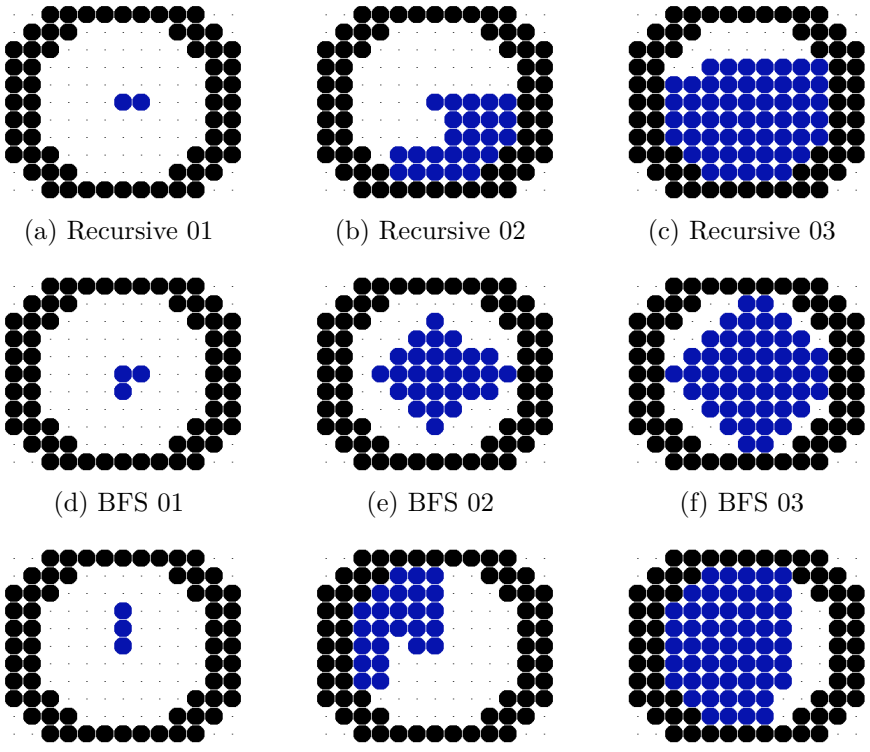
The sequential algorithm achieves a good performance in the low resolution mode, but its frame rate drops significantly in the high definition

mode. This performance decrease is due to the increased number of pixel that need to be processed. The algorithm processes all pixels in the buffer, disregarding the size of the captured body.

The DFS algorithm shows a good performance in the low resolution mode and in the high definition mode respectively. The elements pushed onto the stack are written in the heap memory, which solves the stack overflow problem. Due to its robustness, this algorithm is the preferred choice for all target buffer sizes.

Implementations using the BFS algorithm show severe performance problems. The output video stream results in stutter and long intervals of no rendering at all.

Figure 4.16 shows how the different algorithms traverse an object.



(a) Recursive 01 (b) Recursive 02 (c) Recursive 03

(d) BFS 01 (e) BFS 02 (f) BFS 03

(g) DFS 01 (h) DFS 02 (i) DFS 03

Figure 4.16: The recursive and the DFS algorithm proceed in different directions successively while the BFS algorithm grows circular from the seed point. In all cases, the seed point is placed in the middle of the object.

4.4.2 Unmanaged code

Unmanaged code in C# applications allows the usage of pointers. Pointer operations are faster than array operations, because not boundary checks are performed.

The usage of pointers has to be declared with the keyword `unsafe`. Additionally, the keyword `volatile` is used to declare that a variable can be modified by multiple threads simultaneously and suppresses compiler optimizations [29]. The following code shows the usage for functions and variables respectively. When using pointers within methods, the fixed statement prevents the variable from being shifted within the memory by the garbage collector [29].

```
1 private unsafe volatile void myVariable; // pointer variable
2
3 // usage of pointers within functions
4 public unsafe void process() {
5   fixed (byte* ptrBuffer = <value>) {
6       //pointer variable ptrBuffer can be used within this block
7   }
8 }
```

4.5 Limitations of the prototype implementation

For the prototype it is assumed that the user is right handed. To generalize the writing hand, the depth frame may be used to determine which hand is closer to the collaborative surface.

The output of the manipulated images strongly depends on the sensor. There are two cases to consider:

1. the sensor looses track of all body joints, and
2. the sensor looses track of the right arm joints.

If the sensor looses track of all body joints, there is no body index assigned. Accordingly, all pixel are background pixel and therefore not processed. The result is either a blank screen or jitter in the video stream, depending on the stability of the sensor signal.

In case the sensor looses track of the right arm joints, the frame is processed, but without a manipulation of the user's arm. The resultant frame shows the user without background. If the sensor signal changes steadily from tracked to non-tracked, the output stream is disturbed by alternating manipulated frames.

Furthermore, the performance depends on the used computing machine. The less memory and the less powerful the CPU is, the longer the rendering of the frames take. In this case, the output stream would have a noticeable low frame rate.

Chapter 5

Conclusion and Future Work

Multiple collaboration methods have been examined and analyzed according to several criteria. As stated in Chapter 2, communicative meta-information such as gestures and mimics are important to enhance conversations. The aspect of multiuser support is difficult as multiple objects have to be tracked while eye-contact has to be maintained. The requirements to a remote collaboration system are global availability and affordability, a maximum visibility of the remote user representation, and the transferability to other users. For tracking the user, RGBD cameras such as the Microsoft Kinect v2 sensor are used that fulfill these requirements and offer a broad palette of functionality.

A prototypical setup has been implemented to enhance interaction gestures of the user. The main aim was to imitate a face-to-face meeting to preserve the personal touch even in a remote collaboration system. As a transparent whiteboard with the camera capturing the user from the front is difficult to obtain and to set up, an alternative system has been chosen which uses a videoconferencing system, an interactive whiteboard and an RGBD camera. A setup where the camera is located at the front right of the user, next to the whiteboard, shows perspective disturbances when it is overlaid with the collaborative content. Therefore, the user's extremities have to be manipulated to balance them out. This enhanced gesture visualizations are achieved by showing the whole body of the user and by additionally replacing the user's cursor by either a selectable symbol, a duplicate of the user's own hand or scaling and rotating the arm to the correct position. In all those scenarios, the personal touch of the communication is perceived as the user's body is transmitted as a whole. However, the symbol mode and the hand mode have shown to distract user. As no direct link between the remote user's body and the interaction point exists, users have a hard time to focus on the current interaction and the user's mimics and gestures at the same time. In contrast, the arm extension mode provides a natural representation of the user. The visual attention is drawn to the interaction point if a touch event occurs, and to the visual representation of the remote

user if no interaction event occurs.

Using a flood fill algorithm facilitates the extraction of specific regions of the body. The body joints serve as sound seed points. Especially for detecting overlapping regions (cf. Sec. 4.2.8), a flood fill algorithm is useful.

As a future work, eye-contact could be realized by positioning the camera on the same side of the screen where the remote user is rendered. In this way, the user looks into the direction of the camera and of the remote user at the same time.

Further improvements on the prototype would include support of hardware acceleration by using the GPU for the computation and the rendering stage, to balance the load between CPU and GPU. This could lead to a significant acceleration of the transmitted video frames, especially when rendering the remote user's representation in high-definition. A limiting factor of transmitting high-definition video frames is the bandwidth. Client based transformation approaches could be examined where the CPU intensive image transformations are rendered on the local machine, and only the most important information is transmitted over the internet.

At the moment, the prototype supports the transformation of the right arm of a user. To support both left-handed and right-handed users, the hand that is closer to the screen could be automatically recognized as the interactive hand by its depth value. of writing hand to support both left-handed and right-handed users. The distance of the hand to the collaborative surface could also be used to predict the touch point and therefore start the transformation of the arm earlier.

Multiple users are generally supported in the prototypical application, but the proper positioning of the different remote user representations and the interferences between them due to overlapping images could be subject of further research.

Furthermore, a user study could help to fully explore the potentials and drawbacks of the prototype.

This thesis examined advanced interaction techniques for remote collaboration. It presents possible scenarios for enhanced gesture visualization and provides a basis for further improvements. However, the prototypical implementation already achieves an increased personal connection to the remote collaborator.

Appendix A

System requirements

Microsoft [25, 26] recommends appropriate system components when working with a Kinect v2 sensor as following.

The personal computer should possess

- Windows 8.1 or later as an operating system,
- a built-in USB 3.0 host controller (PCIe 2.0 with IntelRenesas Chipsets),
- a 64 bit processor (x64),
- at least 4 GB memory,
- at least an i7 processor with 3.1 GHz (physical dual-core with 2 logical cores per physical core).

Furthermore, the sensor requires a DirectX 11.0 capable graphics adapter, such as

- Intel HD 4400 integrated display adapter, or
- ATI Radeon HD 5400 series, or
- ATI Radeon HD 6570, or
- ATI Radeon HD 7800 (256-bit GDDR5 2GB1000Mhz), or
- NVidia Quadro 600, or
- NVidia GeForce GT 640, or
- NVidia GeForce GTX 660, or
- NVidia Quadro K1000M.

Concerning the software and libraries, the following components are recommended:

- Visual Studio 2012 Windows Desktop,
- C# .NET WPF Project, and
- Kinect v2 SDK.

Appendix B

CD Content

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

B.1 PDF files

Path: /

RemoteCollaboration.pdf ..... Master thesis

B.2 Source Code

Path: /sourceCode/

MILBoard/ ..... source code of prototype

B.3 Images

Path: /images/

*.jpg, *.png ..... original raster image
*.pdf ..... original Adobe pdf

References

Literature

[1] Wilhelm Burger and Mark J. Burge. *Digitale Bildverarbeitung*. 1st ed. Springer-Verlag, 2005 (cit. on pp. 36–38).

[2] Teresa Farroni et al. "Eye contact detection in humans from birth". In: *Proceedings of the National Academy of Sciences* 99.14 (2002), pp. 9602–9605 (cit. on p. 1).

[3] Edward T. Hall. *The Hidden Dimension*. Doubleday, 1966 (cit. on p. 1).

[4] Donald Hearn and M. Pauline Baker. *Computer Graphics: C Version*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1997 (cit. on pp. 29, 32).

[5] Keita Higuchi et al. "ImmerseBoard: Immersive Telepresence Experience Using a Digital Whiteboard". In: *Proceedings of the CHI Conference on Human Factors in Computing Systems*. Seoul, Republic of Korea: ACM, 2015, pp. 2383–2392 (cit. on pp. 6, 21).

[6] Francis S. Hill Jr. and Stephen M Kelley. *Computer Graphics Using OpenGL*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006 (cit. on pp. 28, 30).

[7] Ellen A. Isaacs and John C. Tang. "What Video Can and Can'T Do for Collaboration: A Case Study". In: *Proceedings of the International Conference on Multimedia*. Anaheim, California, USA: ACM Press, 1993, pp. 199–206 (cit. on p. 1).

[8] 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 Press, 1992, pp. 525–532 (cit. on pp. 2, 4).

[9] Shahram Izadi et al. “Dynamo: A Public Interactive Surface Supporting the Cooperative Sharing and Exchange of Media”. In: *Proceedings of the UIST Symposium on User Interface Software and Technology*. Vancouver, Canada: ACM Press, 2003, pp. 159–168 (cit. on p. 5).

[10] S. Izadi et al. “C-Slate: A Multi-Touch and Object Recognition System for Remote Collaboration using Horizontal Surfaces”. In: *Proceedings of the Tabletop International Workshop on Horizontal Interactive Human-Computer Systems*. IEEE Computer Society, 2007, pp. 3–10 (cit. on pp. 7, 8).

[11] Brett R. Jones et al. “IllumiRoom: Peripheral Projected Illusions for Interactive Experiences”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Paris, France: ACM Press, 2013, pp. 869–878 (cit. on pp. 5, 6).

[12] Brett Jones et al. “RoomAlive: Magical Experiences Enabled by Scalable, Adaptive Projector-camera Units”. In: *Proceedings of the UIST Symposium on User Interface Software and Technology*. Honolulu, Hawaii, USA: ACM Press, 2014, pp. 637–644 (cit. on p. 5).

[13] Shunichi Kasahara et al. “Second surface: Multi-user Spatial Collaboration System Based on Augmented Reality”. In: *Proceedings of the SIGGRAPH Conference and exhibition on computer graphics and interactive techniques in Asia*. ACM Press, 2012, 20:1–20:4 (cit. on p. 5).

[14] Robert Sedgewick. *Algorithms in C, Part 5: Graph Algorithms*. Addison-Wesley Longman Publishing Co., Inc., 2002 (cit. on pp. 37, 38).

[15] 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 p. 4).

[16] Cameron Teoh, Holger Regenbrecht, and David O’Hare. “How the Other Sees Us: Perceptions and Control in Videoconferencing”. In: *Proceedings of the OzCHI Australasian Conference on Computer-Human Interaction*. Melbourne, Australia: ACM Press, 2012, pp. 572–578 (cit. on p. 1).

[17] Cameron Teoh, Holger Regenbrecht, and David O’Hare. “Investigating Factors Influencing Trust in Video-mediated Communication”. In: *Proceedings of the OzCHI Australasian Conference on Computer-Human Interaction*. Brisbane, Australia: ACM Press, 2010, pp. 312–319 (cit. on p. 1).

[18] A. Valadares and Cristina V Lopes. “Virtually Centralized, Globally Dispersed: A Sametime 3D Analysis”. In: *Workshop on Location Awareness for Mixed and Dual Reality (LAMDa’11)*. 2011 (cit. on pp. 7, 8).

[19] Reinhard Viertl. *Einführung in die Stochastik*. Springer-Verlag, 2003 (cit. on p. 39).

[20] Cha Zhang et al. "Viewport: A Distributed, Immersive Teleconferencing System with Infrared Dot Pattern". In: *IEEE Transactions on Multimedia* 20.1 (2013), pp. 17–27 (cit. on pp. 8, 9).

[21] Jakob Zillner. "3D-Board: A Remote Collaborative Workspace Featuring Virtual 3D Embodiments". MA thesis. Hagenberg, Austria: FH Oberösterreich – Fakultät für Informatik, Kommunikation und Medien, 2014 (cit. on p. 1).

Online sources

[22] Inc. CASIO America. 2015 (accessed August 3, 2015). URL: http://www.casio-intl.com/asia-mea/en/projector/whiteboard/ya_w72m/ (cit. on p. 11).

[23] Yinpeng Chen et al. *ViiBoard: Vision-enhanced Immersive Interaction with Touch Board*. 2015 (accessed June 22, 2015). URL: http://research.microsoft.com/en-us/projects/mic_viiboard/ (cit. on pp. 5, 6).

[24] w'inspire GmbH. 2015 (accessed August 3, 2015). URL: https://we-inspire.com/technology (cit. on p. 11).

[25] Microsoft. 2014 (accessed July 3, 2015). URL: https://msdn.microsoft.com/en-us/library/dn799271.aspx (cit. on pp. 13, 44).

[26] Microsoft. 2014 (accessed July 3, 2015). URL: https://www.microsoftvirtualacademy.com/en-us/training-courses/programming-kinect-for-windows-v2-jump-start-9088 (cit. on pp. 13, 15, 44).

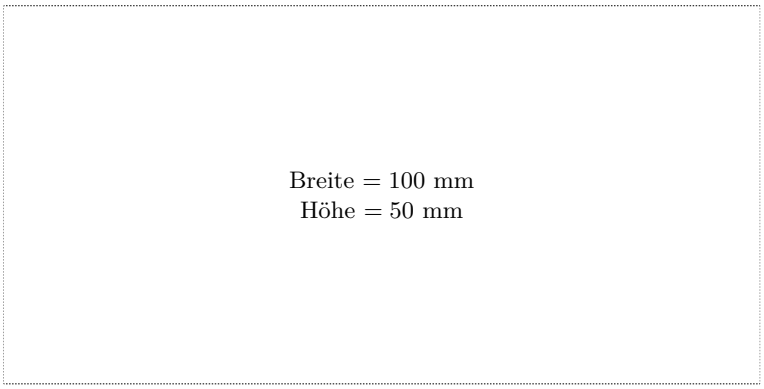
[27] Microsoft. 2015 (accessed August 3, 2015). URL: https://www.microsoft.com/microsoft-surface-hub/ (cit. on p. 11).

[28] Microsoft. 2015 (accessed August 4, 2015). URL: http://download.microsoft.com/download/6/7/6/676611B4-1982-47A4-A42E-4CF84E1095A8/KinectHIG.2.0.pdf (cit. on p. 13).

[29] Microsoft. *C#-Referenz*. 2015 (accessed August 21, 2015). URL: https://msdn.microsoft.com/de-de/library/x53a06bb.aspx (cit. on p. 41).

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —