Collaborative Workspace Awareness in Browser-Based 3D Environments

Christian Elsässer

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im Oktober 2016

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, October 18, 2016

Christian Elsässer

Contents

Declaration **iii**

Abstract **vii**

Kurzfassung **viii**

1 Introduction **1**

2 State of the Art **3**
2.1 3D graphics 3
2.2 Browser-based applications 3
2.3 Browser-based 3D technology 4
2.4 Collaborative workspaces 4
2.4.1 Strict WYSIWIS 4
2.4.2 Relaxed WYSIWIS 5
2.5 Workspace awareness in 3D environments 5
2.5.1 Methods 6
2.5.2 Alternate views 7
2.5.3 Real-life implementations 8
2.6 Workspace awareness in non-3D environments 8
2.7 Related work 9

3 Own Approach **10**
3.1 Environment 10
3.1.1 A browser-based approach 10
3.1.2 Requirements 11
3.2 Methods 12
3.2.1 Visual embodiments 12
3.2.2 Embodiment enhancements 14

4 Technical Design **19**
4.1 Evolution of features 19
4.2 Technologies 19
4.2.1 Meteor 20

4.2.2 MongoDB 20
4.2.3 Three.js 20
4.3 Mode of operation 21
4.3.1 New user connects 21
4.3.2 Download 3D model and texture 21
4.3.3 Request name from user 21
4.3.4 Render 3D scene 22
4.3.5 Update user data in database 22
4.3.6 Fetch other users' data 23
4.3.7 Update 3D scene with other user's data 23

5 Implementation 24
5.1 Initializing the 3D workspace 24
5.2 Importing an object 27
5.3 The user 28
5.4 Toggling the awareness tools 29
5.5 Awareness tools 30
5.5.1 The `animate` function 30
5.5.2 The `updateEmbodiment` function 32
5.5.3 The `insertEmbodiment` function 32
5.5.4 3D cursor 35

6 Evaluation 38
6.1 Methodology 38
6.1.1 Step 1 38
6.1.2 Step 2 38
6.1.3 Step 3 42
6.1.4 Step 4 42
6.2 Aggregated test data 43
6.2.1 Step 2 44
6.2.2 Step 3 45
6.2.3 Step 4 45
6.3 Analysis 46

7 Conclusion 50
7.1 Outlook 50

A Content of the CD-ROM/DVD 52
A.1 PDF-Files 52
A.2 Online Literature 52
A.3 Other 52

References 53
Literature 53

Online sources 53

Abstract

In collaborative applications there is always a loss of information compared to face to face situations. Tools for workspace awareness try to compensate this by providing additional information and are therefore an important aspect of collaborative 3D applications. But especially in the emerging field of web-applications academic insights on collaborative workspace awareness in 3D space are hardly to be found. As a result, the few tools which are available today provide none or, sometimes worse, poorly done workspace awareness tools. This thesis describes existing and, in the course of working on this thesis, refined tools for workspace awareness and evaluates them in a practical way by testing the tools within a collaborative 3D web-application. The results are then considered to build up a basic view on how workspace awareness tools can be utilized in an effective way.

Kurzfassung

In kollaborativen Anwendungen wird, im Vergleich zu persönlichem, gemeinsamen Arbeiten, weniger Information transportiert. Methoden zu einer klareren Wahrnehmung des Arbeitsraumes versuchen das auszugleichen, indem sie zusätzliche Informationen zur Verfügung stellen. Sie sind daher von großer Bedeutung in kollaborativen 3D Anwendungen. Aber besonders im immer wichtiger werdenden Bereich der Web-Applikationen ist die Erfahrung mit Methoden zur Wahrnehmung innerhalb kollaborativer Arbeitsumgebungen noch gering. Daraus resultiert, dass jene wenigen Applikationen welche in diesem Sektor verfügbar sind keine, oder manchmal noch schlechter, unzureichend ausgeführte Hilfsmittel für diese Art der Wahrnehmung bieten. In dieser Thesis werden existierende und im Rahmen der Arbeit weiterentwickelte Hilfsmittel für eine klare Wahrnehmung des Arbeitsraumes beschrieben und auf praktischem Wege getestet, indem sie in einer kollaborativen 3D Web-Applikation Anwendung finden. Unter Berücksichtigung der Ergebnisse dieser Tests wird anschließend versucht, einen Überblick darüber zu geben wie solche Hilfsmittel effektiv eingesetzt werden können.

Chapter 1

Introduction

Nowadays it is common for people to work together despite being geographically separated. Information technologies offer great ways of overcoming this by introducing digital working environments. To successfully communicate remotely it is immensely helpful to know exactly what the other members of a meeting are talking about. The field of 3D graphics is particularly hard to collaborate in online because of the complex nature of the projects. Therefore, a visually representation not only of the scene itself but of everyone viewing and editing it greatly enhances communication. The ability to perceive the other users inside a workspace is called collaborative workspace awareness. Especially in the field of browser-based collaborative 3D workspaces there are seldom any awareness tools available which can be interpreted as a lack of practically tested methods or lack of consciousness for the importance of collaborative workspace awareness tools. The goal of this thesis is to build upon existing methods for collaborative workspace awareness in 3D space, include these methods into a web-application and refine them where reasonable. After a practical test with multiple users the results shall be evaluated and discussed.

Structure of this thesis: The thesis is structured in six chapters. Following this introduction, chapter 2 provides insights on the used vocabulary and introduces the basics of collaborative workspace awareness. Chapter 2 ends with a short description of related work. Chapter 3, covers the concepts behind the testing environment and the implemented awareness methods. Based on the concepts from chapter 3, the following chapter 4 describes the thoughts that went into the structure of the developed application. Chapter 5 provides detailed information of how the application and especially the integrated collaborative workspace awareness methods were implemented. Chapter 6 describes how the evaluation was approached, the data which was gathered and contains an analysis of this data. Last but not least there is chapter 7 which draws some conclusions from the preceding chapters, contains personal thoughts of the author and provides a short lookout of where things could be going in the future.

Chapter 2

State of the Art

The following pages will give an overview of the necessary foundations, current state of the art and exemplary approaches in the field of workspace awareness. Due to the fact that there are very few tools which allow proper collaborative viewing in three-dimensional space, part of the chapter will address related approaches in well-established non-three-dimensional applications. This section will very shortly describe the most important terminology:

- 3D graphics opposed to stereoscopy,
- browser-based 3D technology,
- collaborative workspaces,
- the WYSIWIS concept.

2.1 3D graphics

Nowadays the term 3D can introduce some confusion when not explained properly. In this thesis, when not mentioned otherwise, the term 3D is used for objects and scenes in a three-dimensional space which are rendered onto a standard two-dimensional computer display as it is done in visualizations or video games. Cinematic technologies like, for instance stereoscopy, which are meant to trick the human brain into interpreting depth and perspective in multiple two-dimensional images will not be of any concern.

2.2 Browser-based applications

When designing an application, a common consideration is whether to develop it as a native or browser-based application. Designing an application to run in a web-browser can have unique advantages as that they do not need to be installed on a user's system, are by nature independent from the operating system and already base on an established server client relationship which can be used to store data or outsource performance-heavy tasks to the server. The downside of browser-based applications is of course general performance which is bound to the executing browser and accessibility of hardware as the application runs in a sand-boxed environment.

2.3 Browser-based 3D technology

When it comes to native, browser-based, three-dimensional content there is hardly any relevant alternative to WebGL. By native it is meant that there is no plugin needed for rendering 3D content like there is for technologies as Flash[1], Silverlight[2] or Unity[3]. WebGL is a highly efficient variant of OpenGL targeted towards web-browsers which is widely supported on modern browsers. Multiple JavaScript libraries are available, dealing with the complexity of WebGL to accommodate new users or help with recurring tasks. The full standard, documentation and other useful information can be found on the website of the Khronos Group[4].

2.4 Collaborative workspaces

A collaborative workspace enables multiple users to view or edit data remotely at the same time. Such workspaces are usually categorized as relaxed WYSIWIS[5] or strict WYSIWIS as described by Mark Stefik in [4].

2.4.1 Strict WYSIWIS

Good examples of strict WYSIWIS systems can be found in any screen sharing software. One person, the host, will share his or her screen to one or multiple other persons, the clients. The host is the only one actually interacting with the system while an exact visual representation of data, as seen by the host, is transmitted to the clients. An exemplary visualization can be seen in figure 2.1. Especially for three-dimensional environments a lot of potential is lost by this approach as the visual representation of a 3D space on a computer display is always two-dimensional. So for the clients there is basically no difference between a strict WYSIWIS system with two or three dimensions.

[1]Adobe Flash, formerly Macromedia Flash was one of the first browser-based technologies for advanced multimedia content (http://www.adobe.com/products/flashplayer.html).

[2]Silverlight is a lesser known alternative to Adobe Flash developed by Microsoft (https://www.microsoft.com/silverlight/).

[3]Unity is a technology for browser-based multimedia content focused on gaming (https://unity3d.com/).

[4]https://www.khronos.org/webgl/

[5]WYSIWIS – What You See is What I See.

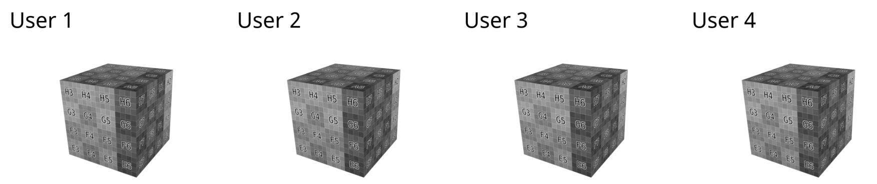


Figure 2.1: Example for strict WYSIWIS. Every user has exactly the same view.

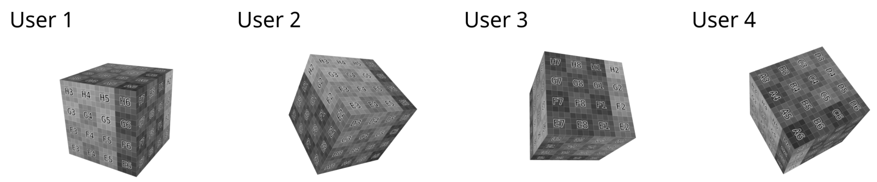


Figure 2.2: Example for relaxed WYSIWIS. Every user sees the same object but has an individual point of view, controlled by himself or herself.

2.4.2 Relaxed WYSIWIS

Contrary to strict WYSIWIS there is usually no host in a relaxed WYSIWIS system. The relaxed variant is much more based on a natural way of collaboration and can be imagined as multiple people sitting together looking at the same object. Still, everybody is seeing the same object, thus the same data but is free to look at it from any direction he or she wants, as exemplary depicted in figure 2.2. Especially in such relaxed WYSIWIS systems it is important to think about workspace awareness as it is not certain that everybody can actually see what one person is referring to from their individual points of view.

2.5 Workspace awareness in 3D environments

A lot of research has already been done in the field of workspace awareness. Especially Carl Gutwin and Jeff Dyck did not only find very interesting methods to establish workspace awareness in three-dimensional spaces, which will be discussed later on, but also found a comprehensive way of describing what workspace awareness actually means. In [2] they suggest to seek the answer for awareness in answering three simple questions:

- Who are the other users?
- Where are the other users?

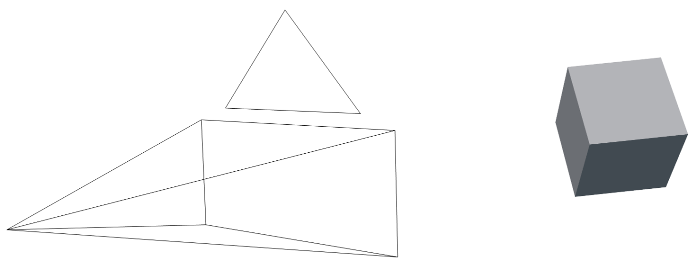


Figure 2.3: Exemplary embodiment. The user is visually represented by a stylized camera.

- What are the other users looking at?

As they further formulate, Gutwin and Dyck see the problem with awareness in the loss of information in virtual systems. Answering those three primary questions in real-life is easy as there is much more data available. Just to name a few, a wide field of view, very good depth perception and localized hearing help to be aware of all the other people in the room. In a virtual system information like this is not available to the human brain, which is why it will lose track of the other users quickly. The solution to this lack of information can be described as simple as providing more information about the surrounding.

2.5.1 Methods

As Carl Gutwin and Jeff Dyck are one of the very few suggesting practical, comprehensible methods, a short introduction about those methods will follow. For further information about those methods see [2].

Embodiments

An embodiment is any kind of visual representation of a user. Usually implemented as a stylized camera, an eye or any other descriptive symbol. An embodiment is visualizing the position in 3D space of the user it represents and can also visualize the direction in which the user is facing, if the implementation allows so. An example for a very simple but efficient embodiment can be seen in figure 2.3.

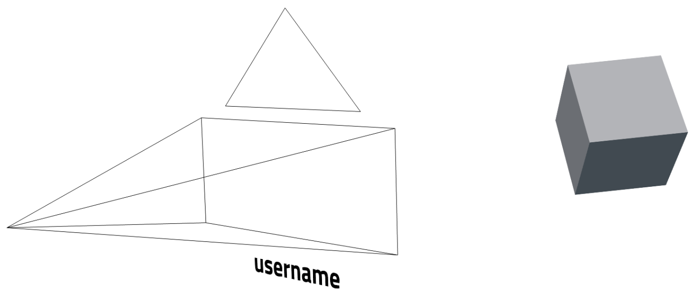


Figure 2.4: Exemplary embodiment enhancement. The user's name is added to his or her embodiment.

Embodiment enhancements

Embodiment enhancements can be any additional information supporting the embodiment. A simple enhancement could be the user's name written next to the embodiment or an additional geometrical element to prolong the embodiment's visibility even if it is off-screen. A very simple example for an embodiment enhancement is depicted in figure 2.4.

Participant list enhancements

A participant list is considered standard in all collaborative applications and therefore not listed separately. But just like any three-dimensional embodiment the list entry can provide additional information. Coloring the name to visualize the activity status of a user is already an enhancement. Enhancement in the participant list can go as far as having a small frame beside the name where the view of this user is rendered in real-time.

2.5.2 Alternate views

In addition to every user's individual view alternate views can be implemented to give a better understanding of the scene as a whole. An aerial perspective or bird's-eye view is comparatively easy to implement and can make a big difference in being aware of the other users. In figure 2.5 an aerial perspective is rendered in the upper right corner to better understand the positions of each object and embodiment. The drawback of alternate views on the other hand is clearly the additional performance when those alternate views are drawn with the same details and objects as the main view, which is necessary in most cases.

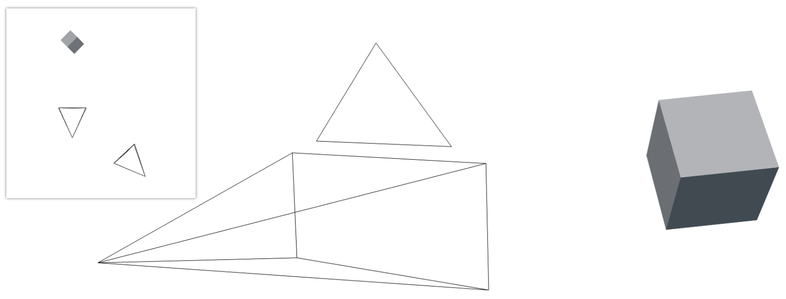


Figure 2.5: Exemplary alternate view as aerial perspective of a scene with a cube and two users, embodied by stylized cameras.

2.5.3 Real-life implementations

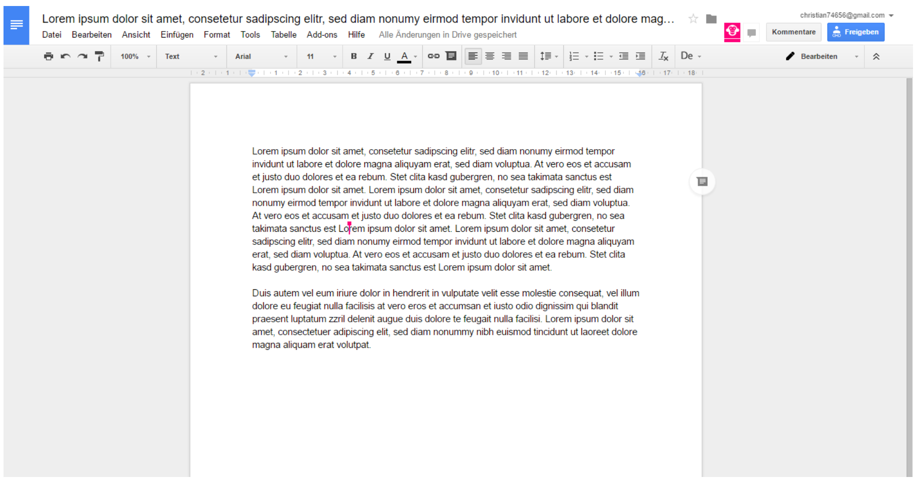
So far there were only very few applications which could be described as real collaborative 3D workspaces. None of them has had enough success to get to a wide audience and as it seems they have been abandoned by their developers. Therefore it is almost impossible to find real life examples of implemented methods for workspace awareness in 3D space.

2.6 Workspace awareness in non-3D environments

In non-3D environments on the other hand there are a lot of collaborative applications, surprisingly most of them browser-based which, intentionally or not, implement methods of workspace awareness quite well. In principle every chat client implements participant list enhancements like a connection status, personal statuses or indicators whether a message has been received or read. The space holding the messages can be seen as the actual workspace, where most chat clients today render the profile picture of the person who posted that message next to it which basically is an embodiment of that user. Some clients also add the users name to that profile picture which then is an embodiment enhancement. More similarities can be found when looking at specific examples.

Google Docs: As there are so many examples of well implemented workspace awareness methods in non-3D applications, I would like to mention Google Docs[6] as it is a relatively complex application, which is nonetheless

[6]Google Docs is a simplified, browser-based alternative to parts of the Microsoft Office suite. Additionally to being free it allows multiple users to edit a file simultaneously. More


Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore mag...
christian74656@gmail.com
Kommentare
Freigeben
Datei
Bearbeiten
Ansicht
Einfügen
Format
Tools
Tabelle
Add-ons
Hilfe
Alle Änderungen in Drive gespeichert
100%
Text
Arial
11
De
Bearbeiten

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Figure 2.6: Google Docs is one of Google's collaborative tools. Embodiment enhancements and user list can be seen in this screenshot.

widely and successfully used by non-experts. When investigating its features, with the principles of Gutwin and Dyck in mind, all three questions can be easily answered by the provided tools. Embodiments, embodiment enhancements and participant list enhancements are tightly integrated and are, intentionally or not, very helpful at being aware of the other users. A screenshot of Google Docs is provided in figure 2.6. Note that the success of the Google Docs application is due to the implementation of those principles but it can for sure be said, that a very example software uses those principles which supports their applicability.

2.7 Related work

A related but quite differently implemented version of some awareness methods, which will be addressed later on, is the Televiewpointer developed by Agustina and Chengzheng Sun. In [1] they present their tool where it is described like a combination of the nose ray, introduced by Dyck and Gutwin and a 3D cursor. The Televiewpointer is implemented on top of CoMaya[7], which is an attempt at bringing collaborative features into Autodesk Maya[8].

information can be found in [5].

[7] http://cooffice.ntu.edu.sg/comaya/

[8] http://www.autodesk.de/products/maya/overview

Chapter 3

Own Approach

This chapter covers the implemented awareness methods including the refinements that went into them as well as the considerations towards the testing environment.

3.1 Environment

It is a well-established practice to develop a testing environment before designing the tools it will contain. This way possible implementation-issues of those tools can be foreseen and appropriate adaptions can be planned beforehand.

3.1.1 A browser-based approach

As mentioned in the title and foregone pages the awareness tools shall be tested against a browser-based environment, i.e., tested in an application which runs in a web-browser. With the rise of web-based applications in the last decade, which is still going on today, the advantages of running an application inside a browser can be experienced in a lot of fields. Where native applications need to be installed, permanently occupy space and are prone to the user not updating the software, a web-application can be used on demand and always in its latest version. Another advantage to be considered is the possibility of using an HTML/CSS user-interface which allows for fast, clean and flexible development. To be considered collaborative the environment should ideally already have a good concept of real-time data exchange established to ease the implementation process. It was clear from the beginning that to reach the goal of a good testing environment multiple users should use the software at the same time. An alternative approach could have been to simulate or record users. Arguably this would have created a more homogenous testing environment with everyone experiencing the exact same situations but would also have taken away the experience of having real users which will react to the test-person's actions. In the end the multi-user approach was chosen to have the user experience a situation close to real life usage at the cost of having slightly different situations.

3.1.2 Requirements

Before implementing or even designing the application the specific requirements it would have to satisfy needed to be defined.

Flexible in number of users

As different group sizes will be testing the awareness methods during evaluation it is of importance for the application to allow users to join and leave a session at any time.

Realtime

Realtime updates are a must have for this environment. Realtime in this context means every change to a dataset from any client needs to be transferred to the server and from there to all other clients within 33 milliseconds resulting in a refresh-rate of 30 fps. Especially server-side JavaScript solutions offer features which can be of great help with this.

3D workspace

As the title says the application will of course need to display three-dimensional data. While the question for a 3D standard technology is still ongoing in native applications surprisingly there is a clear frontrunner in the much younger segment of web-applications. WebGL benefits of quite a few JavaScript libraries which are easy to use and still allow for very deep control where it is needed. This fact as well as its very good performance make WebGL the number one choice for any ambitious web-based 3D project.

Relaxed WYSIWIS

The workspace should honor the relaxed WYSIWIS concept. This makes keeping the data on the server up to date more challenging because updates will be pushed from any client multiple times a second and need to be pushed to the other clients rapidly while ensuring the integrity of the database.

Simple to change and extend

As the implementation will involve some experimenting it is important for the environment to be easily extendable. At this point using a framework and not building the application from scratch seems to be the better option.

Ability to turn the awareness tools on and off

For easier and focused testing all awareness tools should be implemented in a way which allows to turn them on and off during runtime for each user individually.

3.2 Methods

The focus of this thesis lies on the awareness methods which will be evaluated in chapter 6. The presented methods are based strongly on the methods described by Carl Gutwin and Jeff Dyck in [2] but adapted where it seemed reasonably. The next sections will introduce those methods.

3.2.1 Visual embodiments

Visual embodiments represent probably the most important category as most of the other methods will base on some kind of visual embodiment. When looking at implementations of visual embodiments in other applications it quickly comes to mind that the type and style strongly depends on the workspace. Where collaborative text-processing uses a cursor for every user, collaborative spreadsheet software lets every user mark an individual cell. So in successful applications users are visually represented by the same kind of embodiment they are used to see in the non-collaborative versions of those individual applications. But in a 3D suite there is no visual representation of the user as he is operating in ego-perspective mode. The next best thing in 3D applications is likely the camera which can be placed in a 3D scene. Different ways of depicting a camera object can be seen in figure 3.1.

A very interesting detail can be found when looking at the characteristics of the representation. Where Maya uses a relatively detailed object, Blender[1] focuses on a very reduced frustum and Cinema 4D[2] seems to try getting the best of both worlds. In the early development stages for this thesis the visual embodiment of the user was very much on the abstract side with a camera model very similar to Blender. This seemed to be a good solution until the first feedback of people with no 3D design background came back. The feedback was very skeptical and looking back the camera approach was not only confusing for some users but formally no user embodiment. It was an embodiment of the user's perspective. After this conclusion the camera approach was replaced by a more user-oriented design. A stylized head and upper body found much wider acceptance and was instantly recognized by everyone. Both approaches are shown in figure 3.2. Although one could argue that from this embodiment an observer could not even know where the frontside and backside is, I think this should not even be strictly

[1]http://www.blender.org

[2]http://www.maxon.net/de/products/cinema-4d-studio.html

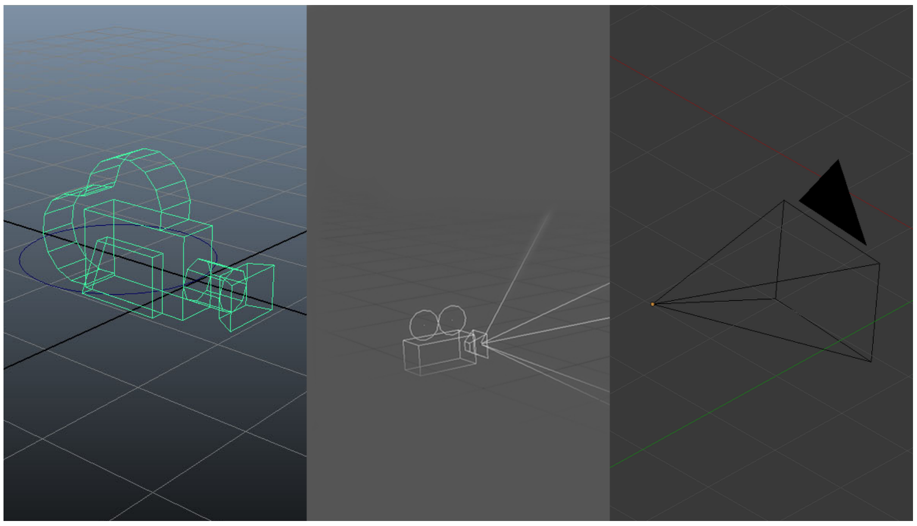


Figure 3.1: Visual representation of camera objects across different applications. From left to right: Maya, Cinema 4D, Blender.

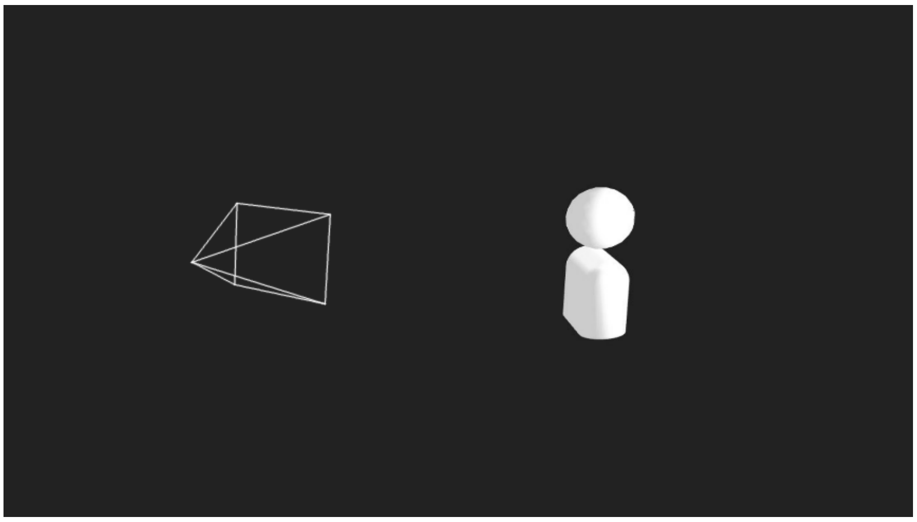


Figure 3.2: On the left the first visual embodiment design strongly oriented by Blender's approach. On the right the latest design which is more focused on the user.

the responsibility of the embodiment but of additional visual aids. This is where embodiment enhancements come in.

This kind of embodiment is very good at depicting a user's position and also at giving an idea about his rotation but only when it actually is currently inside the view port. A very simple way of making sure that the

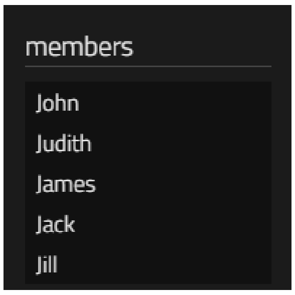


Figure 3.3: A very simple list of users without any enhancements.

amount and identity of present users is always clear at first sight can be a user list. When a user joins a session he simply enters a name and will then be listed on the right hand side of the screen together with all other users currently in this session as seen in figure 3.3.

3.2.2 Embodiment enhancements

In the last paragraphs it was mentioned that a visual embodiment is very important but it has to be enhanced with additional aids to give a good impression about the user it is embodying. It was decided to implement three different embodiment enhancements for different purposes.

Frustum

In the comparison of different camera representations in figure 3.1 can be seen that Cinema 4D tries to combine a camera with arguably unnecessary details with a frustum to enhance the data it carries with the viewing angle. This approach started with the same idea. During development the question came up how big the frustum should be. The viewing angle defined how wide the frustum had to be at any given distance but the size of the frustum can also carry additional information. At last it was decided that the viewing plane should be at the near clipping[3] plane of the camera. This way an observer could see when a user is too close to an object to see it. This can also come in handy when looking inside of objects to see how far inside a user is looking. Last but not least the viewing frustum also indicates in which direction the user is looking. This solves the problem mentioned with the visual embodiment in section 3.2.1 to a point where not only position but also rotation and viewing angle of a user is unambiguous. The advantages of enhancing an embodiment with a frustum can be seen in figure 3.4

[3]Clipping is a term which describes the hiding of objects too near or too close to the camera to prevent display errors and improve performance.

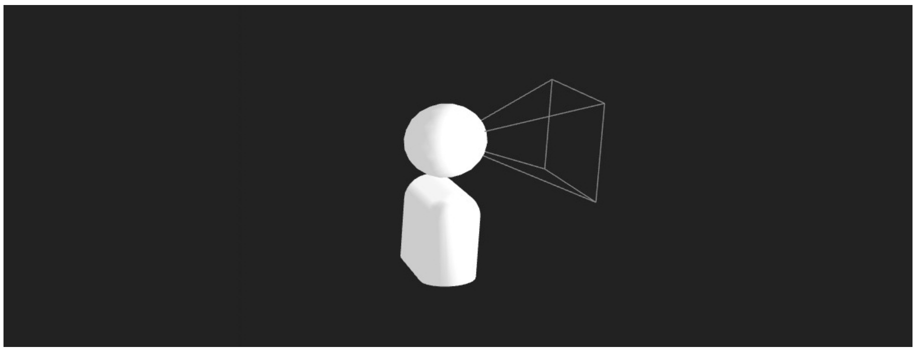


Figure 3.4: The user embodiment is enhanced with a frustum to indicated rotation, viewing angle and near clipping.

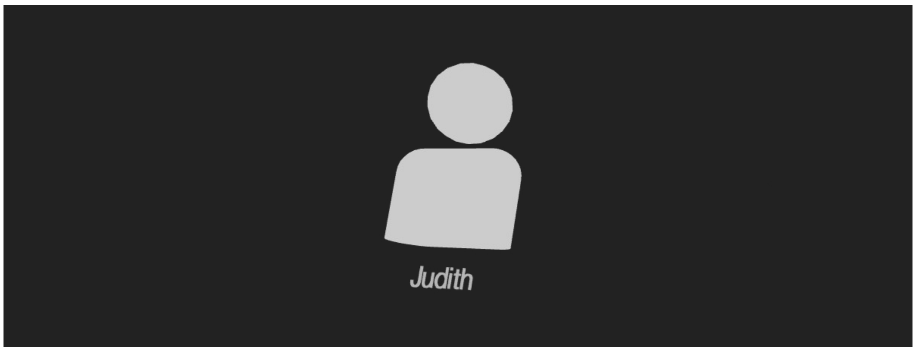


Figure 3.5: The name below the user embodiment identifies the user.

Names

At this point the visual embodiment of a user still lacks any identity. The first possibility which comes to mind when thinking about possible ways of enhancing an embodiment with identity data is simply adding an identifier, in this case the name of the user, to it. After some trial and error, it was decided to display the name as a three-dimensional text element below the embodiment as depicted in figure 3.5 because this way it is obviously visually joined to a single embodiment. When placing it above or beside the embodiment its belonging was not clear once distances between users got smaller.

Nose ray

The concept of a nose ray is to start from the center of the user embodiment and draw a line, i.e., a ray pointing into his viewing direction. In contrast to the frustum, the nose ray is intentionally very long and helps to be aware of users which are not directly visible because they are off-screen or covert by

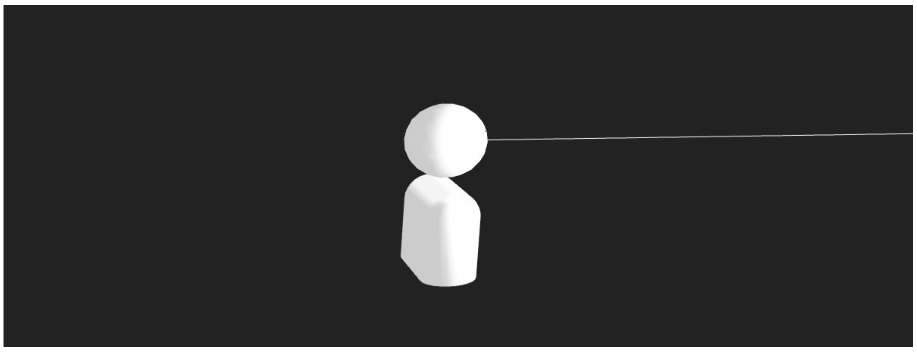


Figure 3.6: An embodiment enhanced with a nose ray.

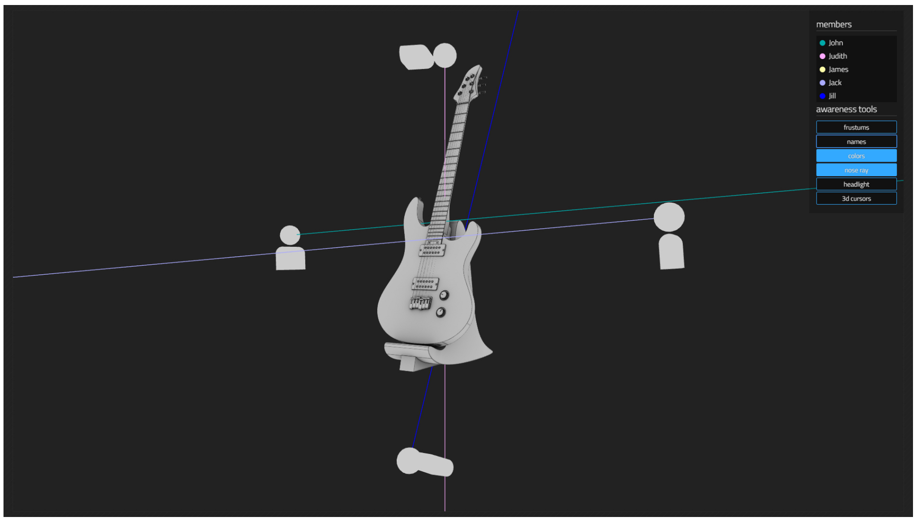


Figure 3.7: Color coding providing fast identification and a clear overview.

other objects. This cannot be accomplished with a frustum as the frustum widens with progressing distance to the user it belongs to. At some point the frustum would be wider than the viewport of an observer and therefore rendered useless. Another advantage of a nose ray is its minimalistic visual footprint which allows to obtain a good and easily understandable overview even with many users. Regarding viewing direction and orientation a nose ray provides all the advantages a frustum provides. The nose ray, as it was implemented, is shown in figure 3.6

Color coding

Color coding is an easy way to correlate different information like the user's name in a list to its 3D embodiment. It is not necessarily bound to the embodiment but can also affect the embodiment enhancements which is why it is not considered a visual embodiment enhancement in this thesis. It does not work without other enhancements or multiple embodiments per user, but rather as an additional awareness aid. For this implementation it was decided to go with coloring only the embodiment enhancements because of the prominent embodiments. This still offers an easy identification at first sight without overdoing the colorfulness of the interface. In figure 3.7 the situation is clear although the nose rays are the only active enhancements. For the implementation it was decided to go for a predefined set of colors which would be assigned randomly to a new user joining a session. Of course this color then has to be locked while the user is active in the session because multiple assignments would render color coding useless.

Headlight

In contrast to Carl Gutwin and Jeff Dyck who describe headlights as an embodiment enhancement in [2], I see it as a separate method for awareness. This is mostly due to the fact that the headlight does not affect the embodiment itself in any way but its environment. The concept behind it is to place a light inside or near the embodiment which will affect the surrounding objects, indicating distance and position inside and outside an observer's viewable area. Additionally, coloring those lights according to a color code can provide information about identity. While in its original concept by Gutwin and Dyck the headlight was described as a spotlight, therefore creating a directional light which would be pointed into the viewing direction of a user it was decided to implement the light in this application as a point-light to emphasize its ability of giving positional information at the cost of losing the information about rotation which on the other hand is already thoroughly displayed through the frustum and the nose ray. See figure 3.8 for the headlight.

3D cursor

The remote 3D cursor tries to display a user's cursor in all other users' 3D viewport. Because of the cursor only being a two-dimensional input it provides no data for the user's local Z axis. This is compensated by a raytracing technique which shoots a ray at the unprojected position of the cursor. If the ray hits an object the 3D cursor-object, basically an unshaded small sphere, for this user is moved to that 3D-position. With multiple positional updates a second, areas and details can be pointed out to other users. An example can be seen in figure 3.9 where the blue user positioned the cursor at the left side of the guitar object's bridge. Other users can see this as a blue dot. The color coded 3D cursor is a good example of how different awareness methods work together as it would require additional visuals like a line connecting it

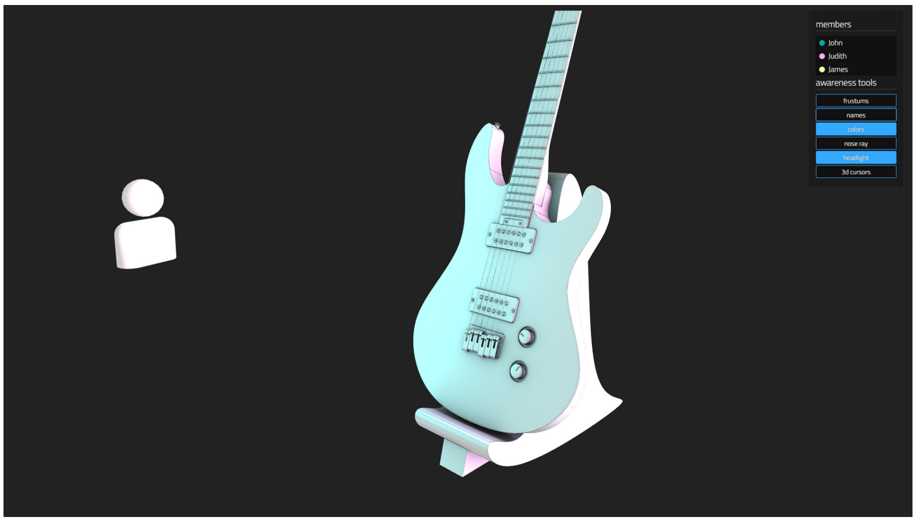


Figure 3.8: A light inside an embodiment lighting the object next to it.

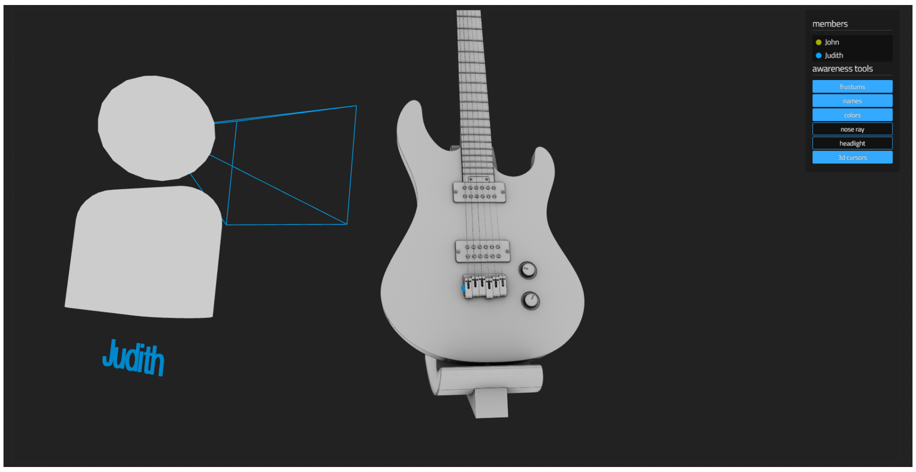


Figure 3.9: The blue dot indicates the user's cursor. Color coding helps to identify to which user the cursor belongs.

to the embodiment to signal its belonging if there would be no color coding.

Chapter 4

Technical Design

This chapter describes the thoughts that went into the structure of the developed application. Section 4.1 contains a short history of how the application evolved. Section 4.2 describes the technologies, which were used to build it and section 4.3 gives insights on the application's internal structure.

4.1 Evolution of features

The testing environment was developed during the third term of the master's degree and continuously improved during the implementation of the awareness methods. The functions of the environment actually decreased during the fourth term. At the beginning it allowed users to register, login, have multiple sessions and invite other users to those sessions. Although those features were already implemented and working it turned out to be more of a distraction than a help to have these features which is why they were ultimately removed from the application. At the time of the evaluations the user management was substituted with a much simpler mechanism which only allowed a newly connected user to enter a name and join one global session. But a detailed look into this will be given in the following paragraphs of this chapter.

4.2 Technologies

The testing environment builds upon established tools to take advantage of well implemented, ready to use functions and save time for the core topic.

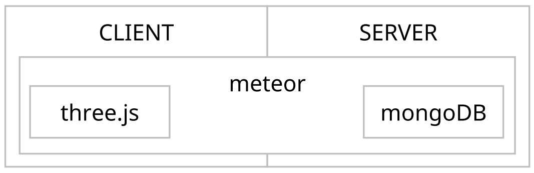


Figure 4.1: Although meteor applications try to blur the line between client and server as much as possible [6] the structure of the test environment needs slightly more separation to work properly.

4.2.1 Meteor

With Meteor[1] it is possible to develop very rapidly. As the same JavaScript code can be run on the server and client and reactive data storage and transfer is already included, Meteor provides valuable features for this application out of the box. Through its asynchronous mode of operation Meteor fits the requirements to a framework for this testing environment very well although the fact that meteor is trying to bring server- and client-code together in a single codebase [6] can have slight impacts on how the application would be structured. This is depicted in figure 4.1 which shows how the connection-spanning nature of meteor is not influencing the strict assignation of Three.js to the client and MongoDB to the server.

4.2.2 MongoDB

MongoDB[2] is a NoSQL database and currently the only system supported by Meteor. Its schema-less approach makes it ideal for fast prototyping and quickly changing requirements. Also performance is good enough for updating and delivering datasets rapidly [7].

4.2.3 Three.js

Three.js[3] is a JavaScript library which wraps most of WebGL's features into comprehensible JavaScript functions and provides useful tools for handling complex 3D data. It covers all the necessary features for this application. A more detailed look at the features of Three.js is given by Michel Krämer and Ralf Gutbell in [3] where they compare performance among different WebGL frameworks.

[1]https://www.meteor.com/
[2]https://www.mongodb.com
[3]http://three.js.org/

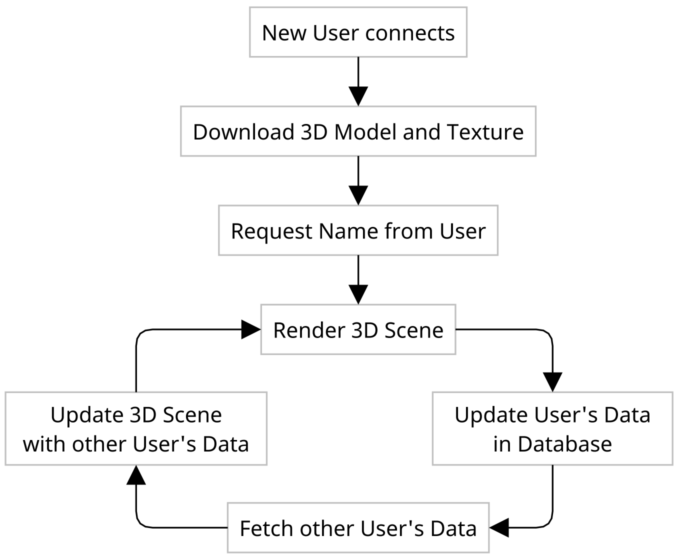

New User connects
Download 3D Model and Texture
Request Name from User
Render 3D Scene
Update User's Data in Database
Fetch other User's Data
Update 3D Scene with other User's Data

Figure 4.2: The key process-steps of the application.

4.3 Mode of operation

The most basic mode of operation can be defined by seven steps which are depicted in figure 4.2.

4.3.1 New user connects

Whenever a user calls the applications URL with a web browser Meteor will establish a websocket connection which will stay connected until intentionally closed by the client. Through this connection all further data can not only be exchanged easily and quickly but it also allows for pushing data from the server to the clients. Once the connection is established Meteor initializes the application's primary function.

4.3.2 Download 3D model and texture

As a first step the 3D model and texture will be downloaded. This happens very early due to the relatively big amount of data. Over 52 megabytes of models and textures are loaded during this step and the application will only proceed when the loading was successful.

4.3.3 Request name from user

At this point the 3D scene is loaded and ready to be rendered. The collaborative functions though stay disabled until the new user enters a new for identification. Although the name is only used for displaying it the other

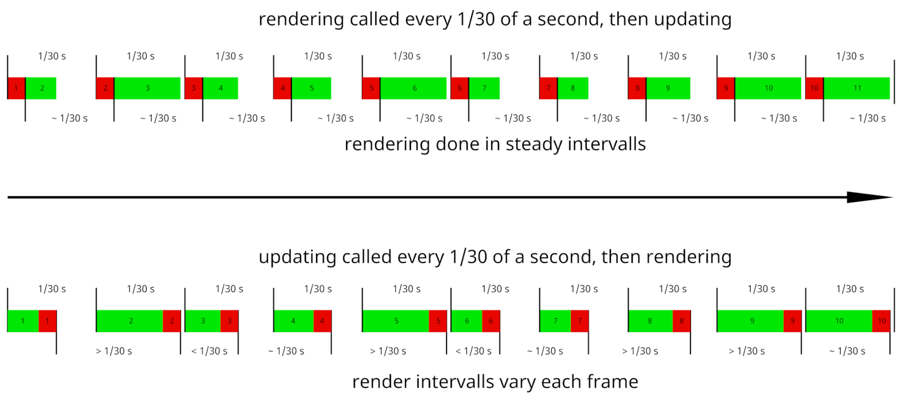

rendering called every 1/30 of a second, then updating
rendering done in steady intervalls
updating called every 1/30 of a second, then rendering
render intervalls vary each frame

Figure 4.3: Rendering the updates of the last frame before calculating the new updates results in a more stable frame-rate as the updates can be of significantly different durations.

users and every client is assigned a unique alpha-numeric ID for programmatic identification the other user's should not be confused with an empty entry in the user-list. Therefore, the application will wait for an input and then go on with the next step.

4.3.4 Render 3D scene

This is where the actual render- and update-cycle will start. The first thing which is done is actually the rendering of the scene although there is no data present of other users at this point. This has two reasons. Firstly, rendering the scene, then updating it with the data for the next frame delivers a significantly more stable frame-rate. Each frame has 1/30 of a second to be updated and rendered. The rendering takes about the same amount of time each frame but updating the data can vary depending on the size of the data and also on the quality of the connection. This principle is demonstrated in figure 4.3. Secondly the render process can fail before any updates are made, which implicitly keeps the database clear from users who cannot render the scene due to hardware or software limitations.

4.3.5 Update user data in database

In the fifth step the user's data will be uploaded to the database, so other clients can access the data for their viewports to render accordingly. The updating is frame-rate-driven and not input-event-driven to not waste performance by writing more frequently than the other clients will render.

4.3.6 Fetch other users' data

Fetch the other users' data, which they will have uploaded in their step 5.

4.3.7 Update 3D scene with other user's data

Display the updated data, create visual representations for new users and delete visual representations when a user disconnected.

Chapter 5

Implementation

In this chapter a detailed look at the implemented features will be given. Insights on the structure of the application and how certain problems were solved during development will be given. Followed by the implementation of the different awareness methods.

5.1 Initializing the 3D workspace

The dominant part of this application is of course the 3D viewport which takes up most of the screen space. In the DOM the viewport is an html5 canvas tag which is drawn to by WebGL. Initializing this viewport is the first thing to do whenever a user enters the application. The JavaScript code for the handling of the 3D data is organized as function which encapsulates all the features of this tool and is subsequently called Tool. Tool is instantiated when Meteor has finished rendering the DOM, which can be detected via the template's rendered function, depicted in the lines 1 to 3, and saved into a global variable for easy access.

```
 1 Template.workspace.rendered = function () {
 2     tool = new Tool();
 3 }
```

When instantiated, Tool calls its own `init` function to setup a Three.js scene. This initialization can be seen in the lines 4 to 36 and is the first function call after the template is rendered in figure 5.1.

```
 4 self.initScene = function () {
 5   self.scene = new THREE.Scene();
 6   self.container = document.getElementById(target);
 7   var WIDTH = self.container.innerWidth;
 8   var HEIGHT = self.container.innerHeight;
 9
10   self.renderer = new THREE.WebGLRenderer({ antialias: true, alpha: true
        });
11   self.renderer.setSize(WIDTH, HEIGHT);
12
13   self.container.appendChild(self.renderer.domElement);
14
15   self.camera = new THREE.PerspectiveCamera(45, WIDTH / HEIGHT, .1, 500)
       ;
16   self.camera.position.set(-6.5, 5.2, 11);
17
18   self.scene.add(self.camera);
19
20   window.addEventListener('resize', self.onWindowResize, false);
21
22   self.controls = new THREE.OrbitControls(self.camera);
23
24   self.controls.target.x = .8;
25   self.controls.target.y = 3.5;
26   self.controls.enableKeys = false;
27   self.controls.update();
28
29
30   setTimeout(function () {
31     self.insertLight();
32     self.insertStaticObject();
33   }, 300);
34
35   self.onWindowResize();
36 };
```

Creating the scene and the renderer (lines 5–11): The first step is to create instances for a 3D scene and a renderer. The scene does not need any explicit configuration while the renderer needs to know the size of the target element so it can work at an appropriate resolution. References to both instances are saved into the variable `self` which is a reference to the current instance of Tool. It is a common pattern to have the variable `self` as a copy of `this` to avoid confusing the contexts as the `this` keyword may refer to something else than the actual instance of Tool in some situations. See [8] for more information on `this` and `self`.

Connecting the scene to the DOM (line 13): The renderer then provides `self.renderer.domElement`, a html canvas element which is appended to the container element in the DOM.

Adding a camera (lines 15–18): To actually see the workspace a camera has to be created and added to the scene. A `THREE.PerspectiveCamera` is created with a viewing angle of 45°, an aspect ratio calculated from the width and height of the container element and clipping limits of 0.1 for the near clipping plane and 500 for the far clipping plane.

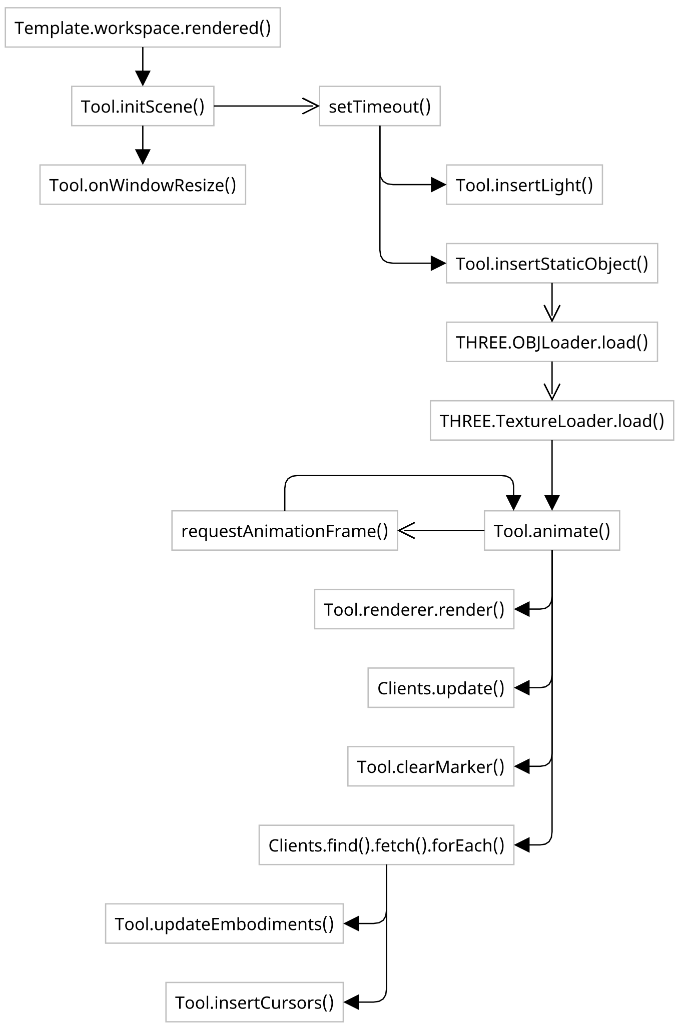


Figure 5.1: The key functions of the application and how they are related.

Making the canvas resizable (line 20): In line 20 a listener on the `resize` event provides the ability to adjust the renderer's resolution and the camera's aspect ratio when necessary.

User controls (lines 22–27): For this kind of application so called orbit controls are common. They allow to orbit around a certain point in 3D space, zoom in and out and also to move this point by panning the view. These kind of controls are instantiated in line 22 and bound to the camera they should control. It is important to call the control's `update` function after the target-point's position has been changed to apply this as the starting position for the user.

Creating the 3D scene (lines 30–33): Now that the scene has been set up, a 3D object is created by calling the `insertObjectByPath` function. This is the object which will be used for the evaluation. Environmental light is added via the function `insertLight`, to provide a basic lighting setup. Both of those steps are called within a `setTimeout` function, so the browser does not block the execution of the script, while the 3D object is loading.

Triggering the `resize` event (line 35): It is a good practice to manually call the `onWindowResize` function at the end of the initialization to ensure correct dimensions for the renderer and camera.

5.2 Importing an object

The scene can hold any 3D object, created on the fly or imported from a file any time, also during runtime. For the test environment it was decided to have a static object loaded from a file to not bother any test-candidate with importing or creating a 3D model. Furthermore, this made it possible to have more ambitious textures and shaders as the parameters could be optimized for this specific model. In the lines 37 to 60 the code for importing the file is shown.

```
37 self.insertStaticObject = function () {
38 var loader = new THREE.OBJLoader();
39 var texLoader = new THREE.TextureLoader();
40
41 loader.load("/obj/rgx-a2.obj", function (data) {
42   texLoader.load('/obj/AO.jpg', function (texture) {
43     var material = new THREE.MeshPhongMaterial({
44       color: 0xeeeeee,
45       map: texture,
46       needsUpdate: true
47     });
48
49     data.children[0].material = material;
50
51     tool.scene.add(data);
52     self.objAssets.push(data);
53
54     self.animate();
55
56     $(".loading_model").fadeOut();
57     $(".choose_name").fadeIn();
58   });
59 });
60 };
```

Line 54 is of special interest as the `self.animate` function is the main rendering function which was not called until now when the 3D model has finished loading.

5.3 The user

When everything is initialized on the client, the server creates a dataset for this user in the database. This dataset, which in MongoDB is called a document, is stored in a collection named `Clients` and holds the user's data. An exemplary dataset can be seen in in the code lines 61 to 78.

```
61 {
62   "_id": "QFst9av9Ni2uxqjps",
63   "pos": {
64     "x": -6.858240149101069,
65     "y": 14.28995154344419,
66     "z": 4.609732437865717
67   },
68   "rot": {
69     "x": -1.1670429774702062,
70     "y": -0.5782619403295505,
71     "z": -0.9073464993323803
72   },
73   "color": 11141290,
74   "colorString": "#aa00aa",
75   "username": "John",
76   "timestamp": 1464119115491,
77   "cursorSurfacePoint": false
78 }
```

While `pos` and `rot` are holding the user's position and rotation in 3D space, timestamp is of particular importance at this point. Because Meteor does not provide a reliable way of knowing when a user disconnects from the server a timestamp is saved and updated on a regular base. In this special case it is updating once a second. Every ten seconds the server checks all timestamps and removes users which have not sent an update for more than five seconds, which is done in the lines 79 to 88. This way it is possible to detect disconnected users and remove them from the scene for better performance and less visual clutter.

```
79 var cleanUpUsers = function () {
80   Clients.find({
81     timestamp: {
82       $lt: ((new Date()).getTime() - 5000)
83     }
84   }).forEach(function (user) {
85     Clients.remove(user._id);
86     Colors.update({ assigned: user._id }, { $set: { assigned: false } })
       ;
87   });
88 }
```

5.4 Toggling the awareness tools

All awareness tools are active by default. As one requirement for the testing environment was to disable and enable them during runtime a set of buttons was implemented which would toggle flags in the `this.awareness` array which can be found in the lines 89 to 108. Those flags are considered during each redraw to determine which features need to be drawn.

```
 89 function Tool() {
 90   this.scene = null;
 91   this.camera = null;
 92   this.controls = null;
 93   this.objAssets = [];
 94   this.objMarker = [];
 95   this.frustums = [];
 96   this.awareness = {
 97     frustum: true,
 98     name: true,
 99     color: true,
100     noseRay: true,
101     headlight: true
102   }
103
104   var self = this;
105   var skipCount = 30;
106
107   ...
108 }
```

It is important to clear the workspace from all embodiments after any change to those toggles, which can be seen in the lines 112, 116, 120, 124, 128 and 132, to enforce a full refresh of all awareness tools on the next redraw.

```
109 "click button": function (e) {
110   switch (e.target.dataset.function) {
111     case "tgl_frustum":
112       tool.awareness.frustum = !tool.awareness.frustum;
113       tool.clearEmbodiments();
114       break;
115     case "tgl_name":
116       tool.awareness.name = !tool.awareness.name;
117       tool.clearEmbodiments();
118       break;
119     case "tgl_color":
120       tool.awareness.color = !tool.awareness.color;
121       tool.clearEmbodiments();
122       break;
123     case "tgl_noseray":
124       tool.awareness.noseRay = !tool.awareness.noseRay;
125       tool.clearEmbodiments();
126       break;
127     case "tgl_headlight":
128       tool.awareness.headlight = !tool.awareness.headlight;
```

```
      tool.clearEmbodiments();
      break;
    case "tgl_3dcursor":
      tool.awareness.cursor = !tool.awareness.cursor;
      break;
  }
  $(e.target).toggleClass("active");
},
```

5.5 Awareness tools

The `animate` function is the primary render function and handles all the functions for drawing the awareness tools. It uses the JavaScript function `requestAnimationFrame` to recursively call itself each frame. This is also shown in figure 5.1.

5.5.1 The `animate` function

In line 138 one can see the recursive call and in line 139 there is the actual render function. Calling the render function at the beginning of `animate` has significant performance advantages as the calculation of the next frame can start immediately. On the other hand, this means that all updates lag one frame behind but this cannot be noticed by the user. In line 141 it is checked whether a valid instance of `Tool` exists and also checks the `skipCount` variable which is used to skip the first 30 frames of calculation. Without skipping the first 30 frames some browsers with weaker performance like Firefox or Safari would sometimes lock up.

Updating the database (lines 144–157): For the visual embodiments the most important thing is to have an up-to-date dataset of each user. This is why one of the first things the animate function does is update the user's position and rotation in the database. At first this was triggered by the mousemove event as soon as the user modified his view but on fast mouse movements this event could be triggered too often for the database to handle without noticeable lag. So it was decided to update this data in the animate function 30 times a second.

```
137 self.animate = function () {
138   requestAnimationFrame(self.animate);
139   self.renderer.render(self.scene, self.camera);
140
141   if (tool && skipCount < 1) {
142     var id = Session.get("id");
143
144     Clients.update(id, {
145       $set: {
146         pos: {
```

```
147           x: tool.camera.position.x,
148           y: tool.camera.position.y,
149           z: tool.camera.position.z
150         },
151         rot: {
152           x: tool.camera.rotation.x,
153           y: tool.camera.rotation.y,
154           z: tool.camera.rotation.z
155         }
156       }
157     });
158 
159     var clientCount = 0;
160     self.clearMarker();
161 
162     Clients.find().fetch().forEach(function (client) {
163       ++clientCount;
164       if (client._id != id) {
165         self.updateEmbodiments(client);
166         self.insertCursors(client);
167       }
168     });
169 
170     var frustumCount = tool.frustums.length + 1;
171 
172     if (clientCount < frustumCount && clientCount > 0) {
173       self.clearEmbodiments();
174     }
175 
176     skipCount = 5;
177   }
178   else { --skipCount; }
179 
180   self.objAssets[0].children[0].material.needsUpdate = true;
181 };
```

Updating the embodiments and cursors (lines 162–168): The functions `updateEmbodiments` and `insertCursors`, which are called for every other user's data, will be explained separately later on.

Clear embodiments when necessary (lines 170–174): If there is a mismatch between the number of clients and the number of drawn embodiments the scene will be cleared from all embodiments. It costs less performance to remove all embodiments and redraw them in the next frame than to distinguish between already drawn and missing embodiments. A missing embodiment can result from a newly connected user whereas a disconnected user would leave a lifeless embodiment in the scene.

Update scene object's material (line 180): The `needsUpdate` property of the scene object's material is set to true, resulting in a recomputation of this material's properties by the next call of `Tool.renderer.render`.

5.5.2 The `updateEmbodiment` function

The `updateEmbodiment` function, shown in the lines 182 to 194 gets called on each frame for each user and updates that user's visual embodiment's position and rotation. If there is no embodiment found for a user it inserts one by calling `self.insertEmbodiment`. The client's data is handed over from the animate function where it has already been fetched.

```
182 self.updateEmbodiments = function (client) {
183   var found = false;
184   self.frustums.forEach(function (frustum, index) {
185     if (client._id == frustum.clientId) {
186       found = true;
187       tool.frustums[index].position.set(client.pos.x, client.pos.y,
        client.pos.z);
188       tool.frustums[index].rotation.set(client.rot.x, client.rot.y,
        client.rot.z);
189     }
190   });
191   if (!found && client.username.length > 0) {
192     self.insertEmbodiment(client._id, client.username, client.color);
193   }
194 }
```

5.5.3 The `insertEmbodiment` function

The call to `insertEmbodiment` in the `updateEmbodiment` function in line 192 creates a new embodiment, adds embodiment enhancements as children, so they translate and rotate with the embodiment, and inserts it into the scene. This is where most of the awareness tools are managed because they relate to the embodiment in some way. As this function is so important, it is explained by each line in the next paragraphs.

```
195 self.insertEmbodiment = function (id, name, color) {
196   if (!self.awareness.color)
197     color = false;
198 
199   var geometry = new THREE.Geometry();
200   var material = new THREE.LineBasicMaterial({ color: color ? color : 0
        xf8f8f8 });
201   var geometry2 = new THREE.Geometry();
202   var material2 = new THREE.LineBasicMaterial({ color: color ? color : 0
        x1199ff });
203 
204   var user = new THREE.Object3D();
205 
206   if (self.awareness.noseRay) {
207     geometry.vertices.push(new THREE.Vector3(0, 0, -100));
208   }
209
210   if (self.awareness.frustum) {
211     geometry.vertices.push(new THREE.Vector3(0, 0, 0));
212     geometry.vertices.push(new THREE.Vector3(-.6, .42, -1.02));
213     geometry.vertices.push(new THREE.Vector3(0, 0, 0));
214     geometry.vertices.push(new THREE.Vector3(.6, -.42, -1.02));
215     geometry.vertices.push(new THREE.Vector3(0, 0, 0));
216     geometry.vertices.push(new THREE.Vector3(-.6, -.42, -1.02));
217     geometry.vertices.push(new THREE.Vector3(0, 0, 0));
218     geometry.vertices.push(new THREE.Vector3(.6, .42, -1.02));
219     geometry.vertices.push(new THREE.Vector3(-.6, .42, -1.02));
220     geometry.vertices.push(new THREE.Vector3(-.6, -.42, -1.02));
221     geometry.vertices.push(new THREE.Vector3(.6, -.42, -1.02));
222     geometry.vertices.push(new THREE.Vector3(.6, .42, -1.02));
223   }
224   else {
225     geometry.vertices.push(new THREE.Vector3(0, 0, 0));
226   }
227
228   var line = new THREE.Line(geometry, material, THREE.LineSegments);
229
230   if (self.awareness.name) {
231     var textMaterial = new THREE.MeshLambertMaterial({ color: color ?
        color : 0xffffff });
232     var text3d = new THREE.TextGeometry(name,{size:.15,height:.01,
        curveSegments:4,font:"helvetiker"});
233     text3d.center();
234     var textObject = new THREE.Mesh(text3d, textMaterial);
235     textObject.position.y = -1.3;
236     user.add(textObject);
237   }
238
239   if (self.awareness.headlight) {
240     var light = new THREE.PointLight(color ? color : 0xffffff, 1, 100);
241     light.position.set(0, 0, 0);
242     user.add(light);
243   }
244
245   var loader = new THREE.OBJLoader();
246   loader.load("/obj/user.obj", function (data) {
247     user.add(line);
248     user.add(data);
249     user.clientId = id;
250     user.position.y = -9999;
251     tool.frustums.unshift(user);
252     tool.scene.add(self.frustums[0]);
253   });
254 };
```

The `insertEmbodiment` function (line 195): The `insertEmbodiment` function expects three parameters. `id`, which is the user's id from the database, the user's name `name` and the user's color `color`.

Determining the color (lines 196–197): Here we can see the consideration of the color coding flag. If it is set to false the `color` variable will be set to false to render everything from now on with the default colors.

Defining the basic assets (lines 199–204): A 3D object, mesh geometries and materials are defined for later usage. The 3D object from line 204 is used as the main object and everything else is parented to that main object.

Creating the nose ray (lines 206–208): If the nose ray flag is set a single vertex is created at position (0,0,-100) which is straight ahead 100 units from the origin of the main object.

Creating the frustum (lines 210–226): If the frustum flag is set, the frustum will be created by manually placing the vertices and adding them to the geometry. If the frustum flag is not set, at least one vertex has to be inserted at position (0,0,0) to provide an endpoint for the nose ray.

Creating a line object (line 228): In line 228 a new `THREE.Line` object is generated, holding all line segments (nose ray and frustum), and saved into the variable `line`.

Creating the embodiment naming (lines 230–237): The user's name is created as a text geometry when the appropriate flag was set. For better readability a new material without specularity is created in line 231. The lines 232 to 233 create the text geometry with the necessary parameters, which is then added to a new text object in line 234.

Creating the headlight (lines 239–243): If the headlight flag was set a new `Pointlight` will be created at the main objects origin. By default, threejs uses one-sided faces so light emitted from a light-object inside a geometry does not get blocked by that geometry. In line 242 the light object is added to the main object.

Add id to main object (line 249): The user's id gets added to the user object so it can be identified further on.

Set main object's position (line 250): The position of the embodiment is set outside the viewable area to ensure it is not visible until the first update with real positional data from the related user.

Save a reference to the main object (line 251): A reference to the main object `user` is added to the frustums array for easier iterating over all embodiments later on. The name frustums is a remnant from an earlier version where the frustum would be saved into an separate array. It would probably be better to call it the 'users' or 'clients' array.

Add main object to the scene (line 252): The object is added to the scene graph and therefore made available for the renderer to display it in the workspace.

5.5.4 3D cursor

The 3D cursor takes a little bit of extra computation and is therefore described separately.

The `mousemove` listener

The 3D cursor has to be computed from 2d data, in particular the screen-coordinates of a user's cursor on the html5 canvas element. To accomplish this, an event listener for the `mousemove` event of the canvas was implemented. This function, shown in lines 255 to 277, uses the function `getSurfacePoint` to calculate 3D coordinates from the X and Y coordinates of the cursor. If a valid 3D coordinate is found it is saved to the database.

```
255 Template.workspace.events({
256   "mousemove canvas": function (e) {
257     var cursorSurfacePoint = tool.getSurfacePoint(e.offsetX, e.offsetY);
258
259     if (cursorSurfacePoint) {
260       Clients.update(Session.get("id"), {
261         $set: {
262           cursorSurfacePoint: {
263             x: cursorSurfacePoint.x,
264             y: cursorSurfacePoint.y,
265             z: cursorSurfacePoint.z
266           }
267         }
268       });
269     }
270     else {
271       Clients.update(Session.get("id"), {
272         $set: {
273           cursorSurfacePoint: false
274         }
275       });
276     }
277   },
278 ...
```

The `getSurfacePoint` function

The `getSurfacePoint` function tries to compute three-dimensional scene-coordinates from two-dimensional screen-coordinates. First the coordinates from the canvas element have to be converted into the right coordinate system. Coordinates in a DOM element have their origin in the upper left corner of an element and will increase from 0 to the width of the element when going right and from 0 to the negative height of the element when going down. Coordinates for a normalized 3D vector to be unprojected need to have their origin in the center with maximums of 1 to the top, 1 to the right, -1 to the bottom and -1 to the left. A vector with the converted coordinates is set in line 284. As there is no Z coordinate coming from the input because of its two-dimensional characteristics the Z portion of the vector will be set to 0 for now.

```
279 self.getSurfacePoint = function (inputX, inputY) {
280   var vector = new THREE.Vector3();
281   var raycaster = new THREE.Raycaster();
282   var dir = new THREE.Vector3();
283
284   vector.set((inputX / self.renderer.domElement.clientWidth) * 2 - 1,
285         -(inputY / self.renderer.domElement.clientHeight) * 2 + 1, 0);
286   vector.unproject(self.camera);
287   raycaster.set(self.camera.position, vector.sub(self.camera.position).
        normalize());
288
289   if (self.objAssets.length > 0) {
290     var intersects = raycaster.intersectObjects(self.objAssets, true);
291
292     if (intersects != false) {
293         return intersects[0].point;
294     }
295     else
296       return false;
297   }
298 };
```

The vector will then be unprojected by the parameters of the camera, which basically is the inversion of the projection, i.e., the process of projecting the three-dimensional scene on a two-dimensional plane to display on a screen. In the lines 287 to 297 the `THREE.Raycaster` module is then used to determine if there is any surface in the way of this ray. If any surface is found the point where that surface is hit by the ray is the three-dimensional point of where the cursor is pointing on the canvas element. If no surface is found, in the way of the ray, the function returns `false`.

Rendering other users' 3D cursors

Rendering the cursors of other users can now easily be done in the `animate` function by clearing all markers in line 160 and redrawing them at their respective new positions in line 166. The reason why markers are cleared and generated every frame, in contrast to the embodiments is that the markers are made of simple spheres. It is faster to recreate such a sphere at a new position than to move the old one according to its new position or remove it when the associated user's cursor is not pointing at a surface in that moment. For the embodiments a full recreation each frame is not faster because the models are loaded from an `obj` file.

Chapter 6

Evaluation

This chapter is about the evaluation of the proposed awareness methods and consists of three parts. Section 6.1 describes the methodology, which was used to aggregate the data. Section 6.2 presents this data and section 6.3 analysis it and draws some conclusion on the results.

6.1 Methodology

The tests involved nine participants who would each complete four steps.

6.1.1 Step 1

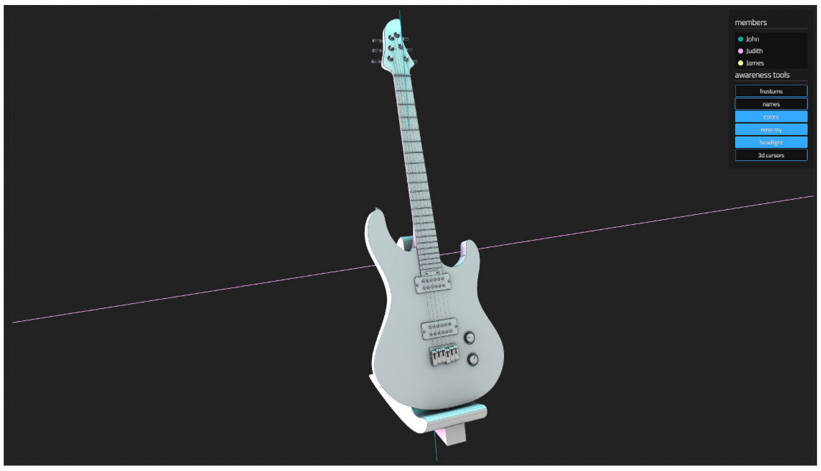
In the first step the participants were introduced into the test environment. As the features to test involved a collaborative component multiple participants took part simultaneously. They were told to make themselves familiar with the environment and the implemented awareness methods.

6.1.2 Step 2

In the second step the users where confronted with ten different situations, each with a question which required the utilization of one or multiple awareness methods. Additionally, the users should state how confident they are about their answers. The following situations and questions were used for step 2.

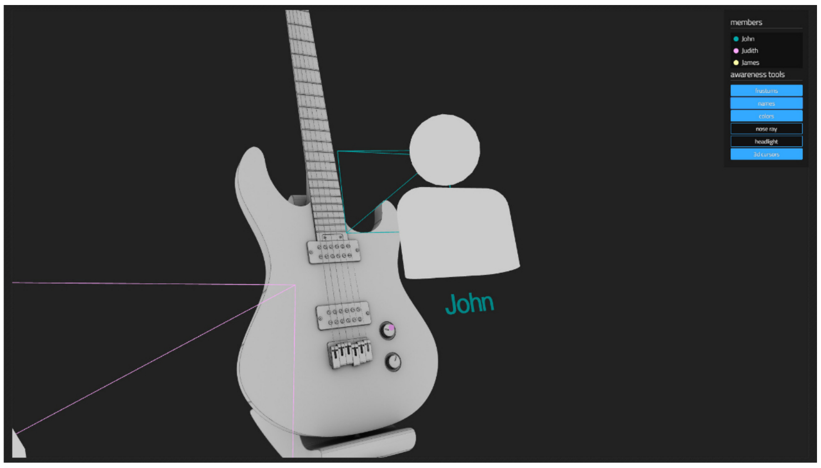
Question 1

The task in question 1 is to find the names of the users in the session and where they are. There were five options with varying names and locations for each user. The situation and question with possible answers are depicted in figure 6.1. The active awareness tools are color coding, the nose ray and the headlight. The last answer was the correct one which can be seen by



No user is directly visible. But where are they?

Figure 6.1: Screenshot of situation 1 with question.



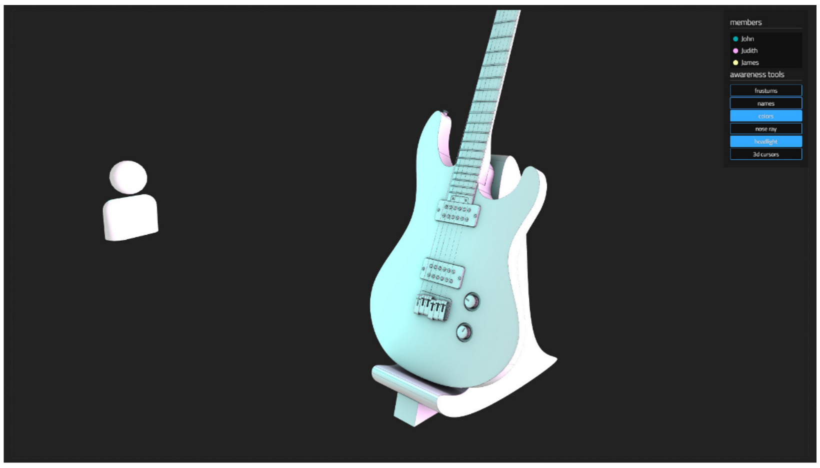
You just joined the Session. John and Judith are already at work. What's happening?

Figure 6.2: Screenshot of situation 2 with question.

the nose rays in the colors of Judith and John. As there is no nose ray for James, the user himself must be James.

Question 2

Question 2 needs the participant to spot a sign of activity in the scene. In this case it was the color coded 3D cursor, depicted in figure 6.2. Three



Which user is visible in the left part of the image?

Figure 6.3: Screenshot of situation 3 with question.

possible answers were given with number 3 being correct.

Question 3

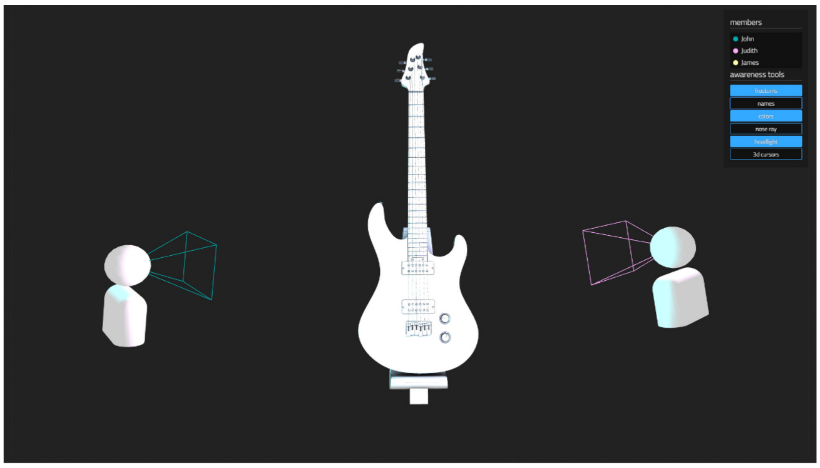
The third situation, shown in figure 6.3 shows a single embodiment which should be identified. The headlight and color coding are active, so the only possibility is to identify the user by those two awareness methods. As there are three users in the session there are three options available. The correct answer is the first one, John.

Question 4

Question 4 asks for the name of the user on the left side of the model. Frustums, color coding and headlights are enabled. The correct answer for this question which is depicted in figure 6.4 is, again, John. Clearly recognizable by the color of the frustum.

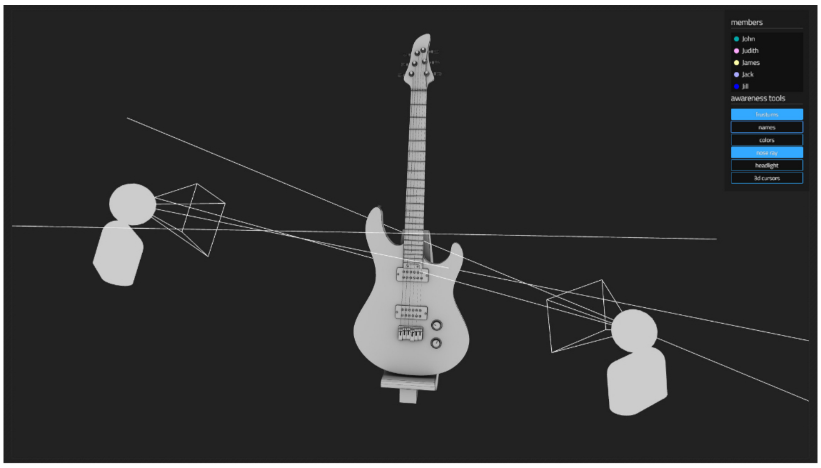
Question 5

To answer the 5th question the participant needs to find the number of users in this session. This can be achieved by counting the nose rays or counting the entries in the user-list. Counting the nose rays in this case can be confusing as color coding is disabled. Question 5 and the according scenario are shown in figure 6.5.



Which user is visible in the left part of the image?

Figure 6.4: Screenshot of situation 4 with question.

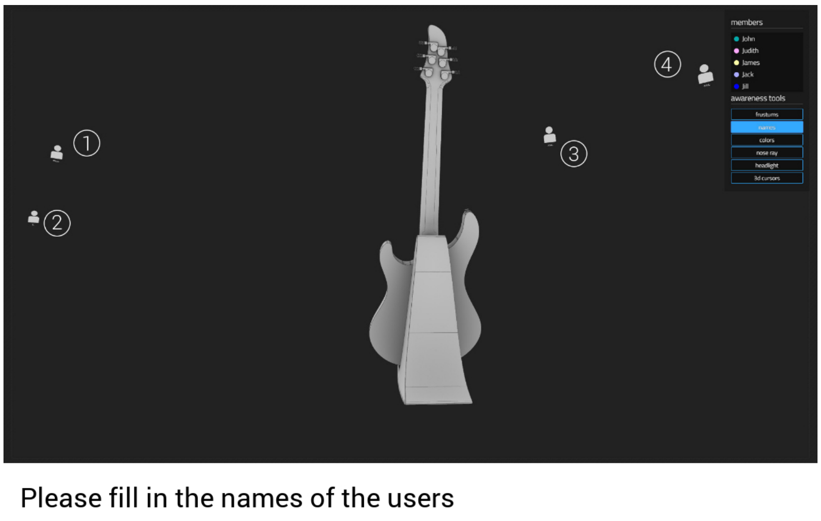


How many users are in this session? On- and off-screen?

Figure 6.5: Screenshot of situation 5 with question.

Questions 6–10

The task in question 6, 7, 8, 9 and 10 is to fill in the names of four users in four text-fields. In all situations the positions of the users and the enabled awareness methods are different. For the scene in question 6 only the naming is enabled (see figure 6.6). Scenario 7, depicted in figure 6.8, shows the users with enabled frustums and color coding while for scenario 8 the headlights and the color coding were enabled which is shown in figure 6.8.



Please fill in the names of the users

Figure 6.6: Screenshot of situation 6 with question.

situation 9 and 10 both have the nose rays active, combined with names in situation 9, depicted in figure 6.9, and color coding in situation 10, depicted in figure 6.10.

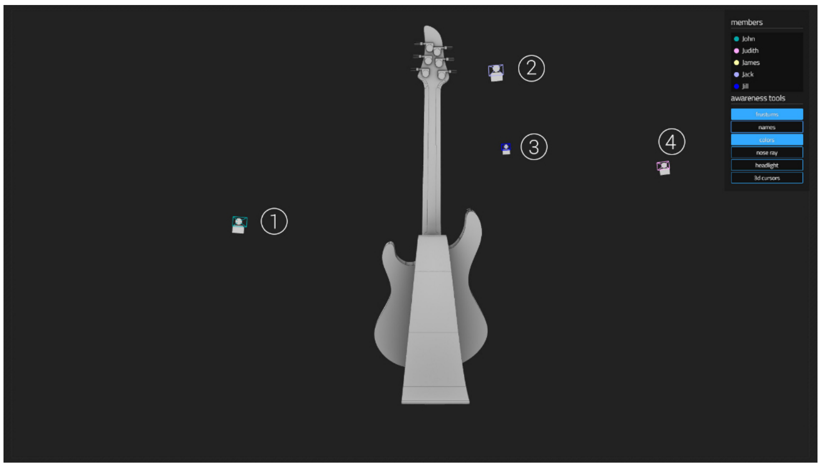
6.1.3 Step 3

In the third step of the test the participants could freely use the test-environment again and, while doing so, were asked to write a few sentences about their impression of the awareness methods. Following four questions were asked and could be answered without limitation in length of the answers:

1. Did you feel you were aware of the other users' position, perspective and identity?
2. Did any of the implemented awareness tools cause particularly positive impressions?
3. Did any of the implemented awareness tools cause particularly negative impressions?
4. Is there anything you would do differently to represent a user's position, perspective or identity?

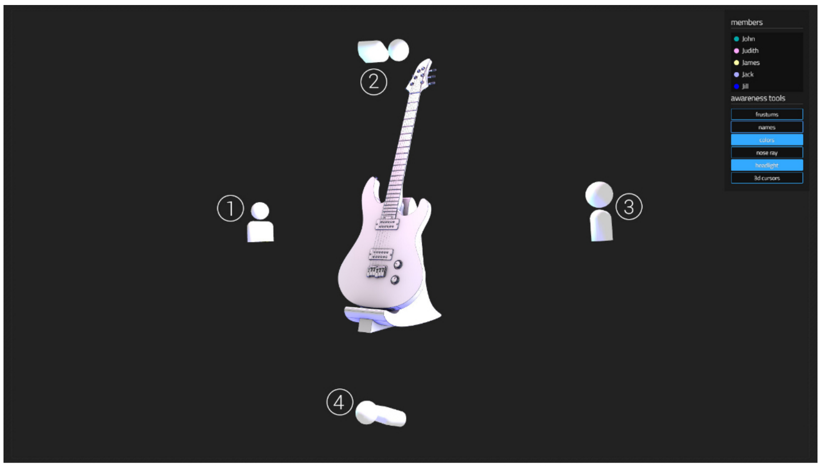
6.1.4 Step 4

In step 4 the participants were asked for their profession, age and whether they had any experience with 3D software or collaborative software to see if there are any correlations between experience and how well the awareness



Please fill in the names of the users

Figure 6.7: Screenshot of situation 7 with question.



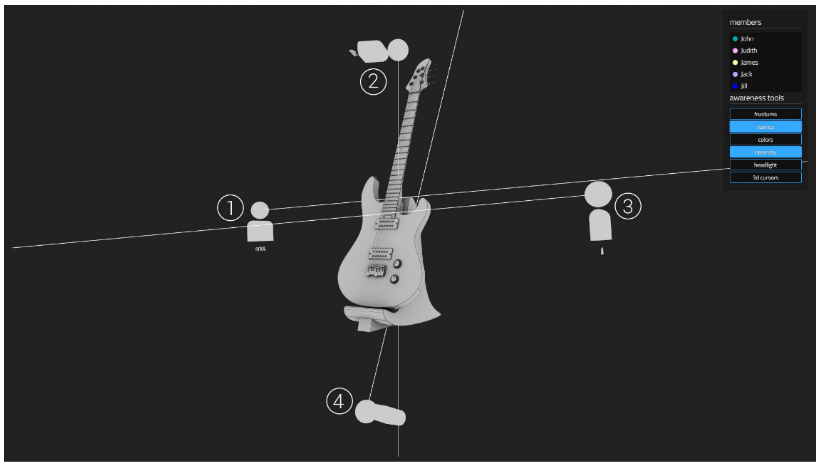
Please fill in the names of the users

Figure 6.8: Screenshot of situation 8 with question.

methods were received. After the fourth step the accumulated data was saved into a JSON file for evaluation later on.

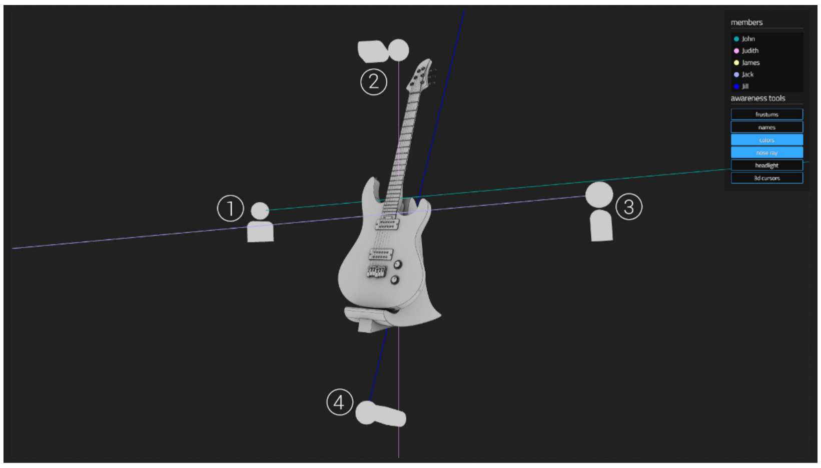
6.2 Aggregated test data

The raw data of all 9 participants is listed here for reference or further analysis. As the first step just served as a way to get familiar with the tools



Please fill in the names of the users

Figure 6.9: Screenshot of situation 9 with question.



Please fill in the names of the users

Figure 6.10: Screenshot of situation 10 with question.

there was no data to be gathered.

6.2.1 Step 2

Table 6.1 shows the outcome of step 2. The first row P1 to P9 represents the participants, the first column Q1 to Q10 the questions as they were described earlier. The other cells contain the answers of the participants. A ✓indicates a correct answer and x indicates an incorrect answer. The digits

Table 6.1: The results of step 2 in tabular form. rows are questions, columns are participants. The correctness of the answers are expressed by a ✓for a correct answer or a x for an incorrect answer. The digits are indicating how sure the user was about his answer.

	P1	P2	P3	P4	P5	P6	P7	P8	P9
Q1	✓(4)	✓(3)	✓(3)	x (4)	x (4)	x (3)	x (3)	x (3)	x (3)
Q2	✓(4)	x (3)	✓(4)	✓(2)	x (3)	✓(3)	✓(1)	✓(2)	x (2)
Q3	✓(3)	✓(2)	x (3)	x(2)	x (3)	x (2)	✓(1)	✓(3)	✓(3)
Q4	✓(4)	✓(4)	✓(4)	✓(4)	✓(4)	✓(4)	✓(3)	✓(3)	✓(4)
Q5	✓(4)	✓(4)	✓(4)	✓(3)	✓(4)	✓(4)	✓(3)	✓(3)	✓(4)
Q6	x (2)	✓(3)	x (2)	x (2)	x (3)	x (1)	x (2)	x (1)	x (2)
Q7	✓(4)	✓(4)	✓(4)	✓(4)	✓(4)	✓(4)	✓(4)	✓(4)	✓(4)
Q8	✓(2)	x (1)	✓(2)	x (3)	x (1)	x (4)	x (3)	x (2)	x (2)
Q9	x (1)	✓(3)	✓(2)	x (1)	x (2)	x (3)	x (3)	x (1)	✓(1)
Q10	✓(4)	✓(4)	✓(4)	✓(4)	✓(4)	✓(4)	✓(4)	✓(3)	✓(4)

Table 6.2: Answers to the first question of the evaluation's step 3.

Question	Did you feel you were aware of the other users' position, perspective and identity?
Participant 1	Most times yes, some tools were more helpful than others
Participant 2	I guess - especially the nose rays and colors where important
Participant 3	Yes
Participant 4	Yes
Participant 5	Yes, in case of colored beams.
Participant 6	Yesssssss
Participant 7	Yes
Participant 8	Yes, really cool :)
Participant 9	Yes

are indicating how sure the user was about his answer and range from 1, the lowest possible level, to 4, the highest possible level.

6.2.2 Step 3

The results of step 3 are shown in table 6.2 for the first question, table 6.3 for the second question, table 6.4 for the third question and table 6.5 for the last question.

6.2.3 Step 4

The professions of the participants were relatively wide spread from an CEO and developers on the one hand to designers and a designated 3D artist on the other hand. The average age of the participants was 29 years and while

Table 6.3: Answers to the second question of the evaluation's step 3.

Question	Did any of the implemented awareness tools cause particularly positive impressions?
Participant 1	coloring works extremely well and at first sight
Participant 2	The names, frustums, colors and nose rays where quite important. They made it clear which user has been on which site of the 3D map
Participant 3	*no answer given*
Participant 4	Runs smoothly
Participant 5	Coloring of the beams und perspektive visional "view-cone" help to find out the other users position and actions.
Participant 6	*no answer given*
Participant 7	colors and nose ray help to identifiy the user
Participant 8	The colors and noseray were very helpful because they were always visible in contrast to the name which is often hidden in some way.
Participant 9	names, frustums and colors were very good

Table 6.4: Answers to the third question of the evaluation's step 3.

Question	Did any of the implemented awareness tools cause particularly negative impressions?
Participant 1	headlight was interesting but useless with more than 2-3 users
Participant 2	the headlights where handy, but if there had been more than 3 users it was quite hard for me to identify them with single users when colors headlights are combined.
Participant 3	not really
Participant 4	No
Participant 5	colorless and nameless characters did not allow to identifiy users and their actions.
Participant 6	No
Participant 7	i wasn't sure about the 3D cursors
Participant 8	I think the headlight will be useless with a lot of members in the scene. It will confuse everyone because it won't be really recognizeable.
Participant 9	3D cursor and headlights

five of nine had experience with 3D software also five of nine had experience with collaborative software.

6.3 Analysis

Judging from the feedback of step 2 and step 3 of the evaluation there are two awareness methods which were perceived extraordinarily positive. The

Table 6.5: Answers to the fourth question of the evaluation's step 3.

Question	Is there anything you would do differently to represent a user's position, perspective or identity?
Participant 1	no
Participant 2	*no answer given*
Participant 3	*no answer given*
Participant 4	No
Participant 5	no, the view-cone and colored action beams tell utmost all. In details the character names are sometimes very small and in the wrong direction (i.e. name reading from right to left).
Participant 6	Different colors
Participant 7	No
Participant 8	No
Participant 9	No

color coding and the nose ray. Not only where they mentioned by multiple participants in a positive way in step 3 but also in step 2 the amount of correct answers was higher when these two methods were enabled. For example, in question 10, where 100% of participants gave the correct answer, with only the color coding and nose rays enabled.

Headlight On the other hand there is the headlight which was considered not very helpful, especially when more than two users were present and made it into four of nine participant's answers regarding particularly negative impressions. Though the feedback was quite critical about the headlight it seemed to work out fine with enabled color coding and not more than two other users in the session as it is indicated by the answers to question 3 of step 2. In question 3 the participants were asked to identify the user on the left with only the headlight and color coding being enabled. 100% of participants gave the correct answer. For future research one could look into enabling the headlight based on the distance between users or users and assets as it definitely proved to be of value under certain circumstances.

Frustums The frustums were mentioned four times in a positive way, two users described them as very important. Regarding the results of step 2 the questions involving a frustum were answered correctly 33 of 36 times.

Color coding As already suggested in the chapter 4 the color coding is a special case of awareness tool as it does not work without anything else being enabled. As some participants of the tests mentioned in their comments this is also a valid point the other way around where some awareness methods are less useful, unusable or even distracting without color coding. This is especially true for the headlight but also for the nose rays. One of the participants would have preferred different colors though it is not stated if this was due to the colors being too similar and therefore hard to distinguish or just for visual reasons.

Names Names have been a more controversial topic among the participants. While some described names as a vital tool for identification, others saw problems in the fact that the text often was not readable because it was heading into a different direction, was too far away or hidden behind the embodiment it should have identified. The answers to the questions in step 2 support the more critical view on names as only 37% of the answers given to questions which involve naming were correct. Retrospectively the positioning of the names worked very well as no participant seemed to have problems with matching the name to the appropriate embodiment but the rotation of the name could be improved by heading it towards the user's viewpoint. Of course this can not be done in a naive way as constraints have to be considered where the text would intersect with the embodiment.

3D cursor The cursor found less attention in the answers of step 3. One participant stated that he was not sure about the 3D cursor another one just listed it in his answer for the question about particularly negative impressions and a third participant described it as "handy" but also mentioned that the headlight would likely be useless with more than three users. In step 2 there was only one question involving the 3D cursor. It was answered correctly by six of the nine participants. The 3D cursor is probably the awareness method with the most possibilities to improve. One idea was to make the size of the cursor dependent on its speed so that a fast movement would increase the cursor's size and a slow movement would decrease it. That way an observer would not lose track of fast cursors but a user would still be able to point out small details when the cursor get's smaller again.

Experience Surprisingly having experience with 3D software or collaborative applications does not seem to have much of an influence on the results of step 2. The most and the fewest correct answers were given by participants with both, experience in 3D software and experience in collaborative software. The two participants without experience in either of those two fields achieved five and six correct answers which is only slightly beneath the average of 6.2 correct answers.

Confidence The participants were a lot more confident when answering questions correctly than they were when giving an incorrect answer. On the scale of 1 "very unsure" to 4 "very sure" the participants stated an average confidence level of 3.375 for correct answers and 2.352 for incorrect answers which suggests that although the participants were not able to answer some questions correctly they were quite aware of which questions they could answer correctly and which not.

General awareness In the first question of step 3 the participants were asked whether they felt aware of the other users' position, perspective and identity. Six of nine participants answered with a clear yes, the remaining three stated that depending on the enabled tools they felt aware of the other users.

Observations During the tests the participants engaged quickly with the awareness methods. Although some participants were skeptical at first, especially those without any prior experience, they understood and used the awareness tools quickly and managed to find their way around the 3D scene and the other users inside it.

Chapter 7

Conclusion

Concluding it can be said for sure that collaborative workspace awareness is a very complex topic with still many open questions to be answered. A single research like this can only slightly touch that topic. Neither the less some interesting results were won in the course of this thesis. With the help of the user-tests it was possible to evaluate the effectiveness of some methods for workspace awareness and spot strengths and weaknesses. Some of the results were foreseeable like the color coding which was considered vital by a majority of testers. Some other results however were quite surprising for me as I expected the nose ray to be less successful. There is no doubt that this field can still provide content for a lot of research. Just testing through all suggested awareness methods in the available literature and drawing a conclusion from those tests could keep teams of researchers busy. For me the process of working on this thesis and the related project turned out to be quite different from what I was expecting. When I started I thought developing a collaborative 3D workspace would take a lot of effort and although I was aware that the tests would be work-intensive I did not expect the actual implementation to be so much easier than to find appropriate test scenarios. I am sure anyone who decides to dive into the topic of workspace awareness in collaborative applications will find reading this thesis at least interesting. Especially as most other research in this field is quite old and most of those papers do not make their test data publicly available.

7.1 Outlook

Although it may take years to see any research in this field in a widely used product I am confident that the field of collaborative 3D software is a growing one. There are constantly tools being developed for this purpose, some of them more successful than others. And even though the focus of this thesis lied on the tools for collaborative workspace awareness another interesting aspect is how relatively easy it was to implement a well working and reliable collaborative 3D workspace. If it is not for the awareness methods themselves maybe the concepts behind the implementation can help others to refine their own implementations and make improvements to the idea of collaboration in 3D space. Especially the combination of Three.js and meteor went surprisingly smoothly considering that these two relatively young technologies have not been used a lot together in the past. I could imagine to see meteor and Three.js being used together more often as developers start to discover the possibilities. Especially because no major problems occurred during development a reorientation towards web-applications could actually make 3D workspaces in browsers a real alternative not only for collaborative software. Web-applications and native 3D graphics are definitely coming together more and more.

Appendix A

Content of the CD-ROM/DVD

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

A.1 PDF-Files

Pfad: /

_DaBa.pdf	Thesis

A.2 Online Literature

Pfad: /literature-online

*.pdf	Copies of online literature

A.3 Other

Pfad: /images

*.eps	Vector graphics
*.jpg, *.png	Raster graphics

References

Literature

[1] Agustina and Chengzheng Sun. "Televiewpointer: An Integrated Workspace Awareness Widget for Real-time Collaborative 3D Design Systems". In: *Proceedings of the 16th ACM International Conference on Supporting Group Work*. GROUP'10. Sanibel Island, Florida, USA: ACM, 2010, pp. 21–30 (cit. on p. 9).

[2] Jeff Dyck and Carl Gutwin. *Awareness In Collaborative 3D Workspaces*. Tech. rep. Department of Computer Science, University of Saskatchewan, 2002. URL: http://hci.usask.ca/publications/2002/groupspace.pdf (cit. on pp. 5, 6, 12, 17).

[3] Michel Krämer and Ralf Gutbell. "A Case Study on 3D Geospatial Applications in the Web Using State-of-the-art WebGL Frameworks". In: *Proceedings of the 20th International Conference on 3D Web Technology*. Web3D'15. Heraklion, Crete, Greece: ACM, 2015, pp. 189–197 (cit. on p. 20).

[4] M. Stefik et al. "WYSIWIS Revised: Early Experiences with Multiuser Interfaces". *ACM Transactions on Information Systems* 5.2 (Apr. 1987), pp. 147–167 (cit. on p. 4).

Online sources

[5] *Google Docs*. URL: https://docs.google.com/ (visited on 06/15/2016) (cit. on p. 9).

[6] *Meteor's client and server architecture*. URL: https://www.discovermeteor.com/blog/what-goes-where/ (visited on 06/15/2016) (cit. on p. 20).

[7] *Performance Testing MongoDB 3.0*. URL: https://www.mongodb.com/blog/post/performance-testing-mongodb-30-part-1-throughput-improvements-measured-ycsb (visited on 06/15/2016) (cit. on p. 20).

[8] *Utilizing a self variable in JavaScript.* URL: http://www.javascriptkata.com/2007/05/14/how-to-use-the-self-with-object-oriented-javascript-and-closures/ (visited on 06/15/2016) (cit. on p. 25).

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —

Breite = 100 mm
Höhe = 50 mm

— Diese Seite nach dem Druck entfernen! —