communicated by:
Lehr- und Forschungsgebiet Informatik 9

Prof Dr. Ulrik Schroeder

The present work was submitted to Learning Technologies Research Group
Diese Arbeit wurde vorgelegt am Lehr- und Forschungsgebiet Informatik 9

Technology-Enhanced Learning of Computer Graphics Rendering Pipeline in Virtual Reality

Technologie-gestütztes Lernen in Virtual Reality am Beispiel der Rendering Pipeline

Master-Thesis
Masterarbeit

presented by / von

Görzen, Sergej
375348

Prof. Dr.-Ing. Ulrik Schroeder

Prof. Dr. Leif Kobbelt

Aachen, November 27, 2020

Contents

List of Figures iii

List of Tables v

List of Listings vi

Glossary vii

1 Introduction 3
- 1.1 Motivation . . . 3
- 1.2 Hypothesis . . . 4
- 1.3 Thesis Goal . . . 4

2 State of the art 5
- 2.1 Scientific Background . . . 5
  - 2.1.1 Technology-Enhanced Learning . . . 5
  - 2.1.2 Virtual Reality . . . 7
  - 2.1.3 Computer Graphics . . . 11
- 2.2 Designing an Educational VR Application . . . 14
  - 2.2.1 Design-based Research . . . 14
  - 2.2.2 Iterative Design . . . 15
  - 2.2.3 Metaphors and Techniques in VR . . . 16
- 2.3 Related Work . . . 16
  - 2.3.1 Scientific Approaches . . . 16
  - 2.3.2 Market Approaches . . . 17
  - 2.3.3 Comparison of Related Approaches . . . 19

3 Conceptual Approach 20
- 3.1 Overall Vision and Objective . . . 20
  - 3.1.1 Visions . . . 20
  - 3.1.2 Objectives . . . 21
- 3.2 Use Cases . . . 21
  - 3.2.1 Stakeholders . . . 21
  - 3.2.2 Personas . . . 22
  - 3.2.3 User Stories . . . 25
  - 3.2.4 Learning goals of the Guided Tour . . . 26
  - 3.2.5 Storyboards of Learning Scenarios . . . 27
- 3.3 Requirements . . . 29

4 Technical Background **30**
4.1 Hardware . . . 30
4.2 Software . . . 31
4.3 Used Technologies and Frameworks . . . 33

5 Realization **37**
5.1 System Overview . . . 38
5.2 Workflow and Communication . . . 38
5.3 Educational VR Application: Rendering Pipeline . . . 38
5.3.1 Namespace EducationalVR . . . 39
5.3.2 Stage 3D Geometry . . . 41
5.3.3 Stages 3D Transformation and Clipping . . . 43
5.3.4 Scene: Guided Tour . . . 47
5.4 Extending: LA Data Collector for VR . . . 50
5.4.1 Web Socket Server . . . 50
5.4.2 Researcher's Configuration Panel . . . 53
5.4.3 VR Data Collector Unity . . . 56
5.5 xAPI Definitions Fetcher . . . 57
5.5.1 xAPI Definitions . . . 58
5.5.2 Fetcher Tool . . . 58
5.5.3 Strict xAPI Statements Showcase . . . 62
5.6 Implemented xAPI Statements . . . 65

6 Evaluation **66**
6.1 Technical Evaluation . . . 66
6.2 Pilot Study . . . 71
6.2.1 Methods . . . 71
6.2.2 Results . . . 73

7 Future Work and Conclusion **76**
7.1 Future Work . . . 76
7.1.1 Rendering Pipeline Prototype . . . 76
7.1.2 Educational VR Stack . . . 78
7.2 Conclusion . . . 79

Appendix **80**

A Bibliography **82**

B Tables **88**

C Listings **92**

D Figures **94**

E Questionnaire Results **101**

F Digital Appendix **104**

List of Figures

0.1 Abstract Cover: Frustum in VR 1

2.1 Learning Analytics Process 7
2.2 Relation between VEE types and degree of realism 9
2.3 Relation between VR worlds classes to degree of interaction 10
2.4 Basic model of Rendering Pipeline 12
2.5 Orthographics and perspective projection 13
2.6 Simplified Rendering Pipeline Model 14
2.7 Design research cycle 15
2.8 High-level iterative process for VR development 15
2.9 Related market approaches 18

3.1 Storyboard: 3D Geometry 28
3.2 Storyboard: 3D Transformation and Clipping 28

4.1 HTC Vive Pro Eye equipment 30
4.2 Unity Editor User Interface 32
4.3 Unity Timeline window 36

5.1 System and stack overview 37
5.2 Communication sequence between clients 39
5.3 Default learning environment 40
5.4 Triangle interaction modes 41
5.5 Changing frustum values with interactions 43
5.6 Perspective projection matrix P 44
5.7 Realization of clipping 46
5.8 Render result as 2D image plane in VR 46
5.9 Space Robot Kyle and dialog panel 49
5.10 Tour director and signals 49
5.11 Guided Tour visualizations 49
5.12 Abstract API data model 52
5.13 Frontend Nuxt.js Class diagram 54
5.14 Researcher's configuration panel - Startpage 55
5.15 Researcher's configuration panel - Dashboard 55
5.16 xAPI Definitions Fetcher Activity 59
5.17 xAPI definitions file hierarchies overview 60
5.18 xAPI Definitions Usage Auto Completions 63
5.19 Create xAPI statement 63

5.20 Create xAPI Statement with context extension 64
5.21 Create xAPI Statement with extensions 64
5.22 xAPI Definitions Fetcher Unity Window 64

6.1 Frustum Transformation Evaluation 67
6.2 Preview of running tests with *mocha* for web socket server. 68
6.3 Preview of running unit tests with *NUnit* 69
6.4 Dashboard with charts of the pilot study in Learning Locker 71
6.5 Pictures of the pilot study. 73

7.1 Transformation matrix preview 77

List of Tables

2.1 Comparison of related approaches 19

3.1 Persona: Pupil 23
3.2 Persona: Student 23
3.3 Persona: Teacher 24
3.4 Persona: Researcher 24
3.5 Persona: Developer 25
3.6 User Stories 26

4.1 System requirements of HTC Vive Pro 31

5.1 xAPI Composition Overview 65

List of Listings

4.1 xAPI statement example 34

5.1 Vertex perspective transformation via Unity script 45
5.2 Code snippet of new socket handling 50
5.3 Old templates CSV columns 51
5.4 New template format in JSON 51
5.5 Generated verb interacted 60
5.6 Generated context virtualReality 61
5.7 Generated context extensions for context virtualReality 61
5.8 xAPI_Definitions Class 61

6.1 Unit test of web socket event "clients:scenario" 68

C.1 xAPI statement complete example 92
C.2 xAPI verb interacted 93
C.3 Code snippet of old socket handling 93

Glossary

AR	Augmented Reality.
CG	Computer Graphics.
CS	Computer Science.
CSV	Comma-separated values.
FOV	Field Of View.
IRI	Internationalized Resource Identifiers.
JSON	JavaScript Object Notation.
LA	Learning Analytics.
LRS	Learning Record Store.
MR	Mixed Reality.
REST	REpresentational State Transfer.
RP	Rendering Pipeline.
TEL	Technology-Enhanced Learning.
VEE	Virtual Educational Environment.
VR	Virtual Reality.
VRLE	Virtual Reality Learning Environment.
VW	Virtual World.
xAPI	Experience API.
XR	Extended Reality.

Abstract

Learning Technologies bring high potential to rethink the traditional ways of learning and teaching. One encouraging technology is Virtual Reality. Its nature to perceive a third dimension creates the possibility to design learning material from other perspectives. Interactive tasks increase motivation and attention, and helps to memorize the acquired knowledge longer. This work investigates the Computer Graphics Rendering Pipeline as learning content for VR and deploys a pilot prototype in Unity, to discover it from another perspective (figure 0.1). After all, the vision of this work is to let a learner perceive the transformation progress of 3D geometry into the resulted 2D images directly inside a Virtual Reality environment, while changing experimental factors to observe real-time changes. This progress is designed in form of a Guided Tour.
To analyze tasks of this learning scenario in VR, all learner actions are logged by xAPI and examined in Learning Locker. This thesis provides background knowledge and documents design decisions for the developed Educational VR Application and its connected systems to support Learning Analytics. This includes the extension of a data collector solution for Virtual Reality and the development of a tool to generate C# classes to ensure consistent composition of xAPI statements inside of Unity. Further in context of this work a pilot study was conducted to identify technical and usability issues and to achieve first findings about its' didactical value.

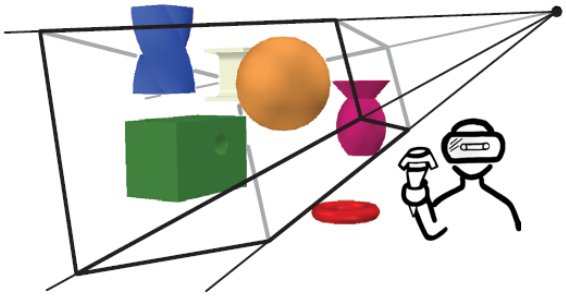


Figure 0.1: Frustum in VR. Frustum sketch was picked from [1].

Zusammenfassung

Lerntechnologien bringen ein großes Potential mit sich und ermöglichen das Überdenken traditioneller Wege der Bildung. Eine vielversprechende Technologie ist Virtual Reality (deu. "Virtuelle Realität"). Durch die Darstellung einer dritten wahrnehmbaren Dimension, werden die Möglichkeiten vom Design von Lernmaterialien erweitert. Interaktive Aufgaben erhöhen die Motivation, Aufmerksamkeit und unterstützen das längere Behalten des Gelernten. Diese Arbeit untersucht die Computer Graphics Rendering Pipeline als Lerninhalt für VR and liefert einen ot-Prototypen in Unity, um diese aus einer anderen Perspektive zu betrachten (Abbildung 0.1). Im Grunde ist die Vision der Arbeit einem Lernenden den Transformationsprozess von 3D Geometrie zu dem resultierenden 2D Bild innerhalb der virtuellen Realität betrachten zu lassen, während dieser in der Lage ist, Faktoren während der Laufzeit zu ändern und die Folgen zu beobachten. Dieser Prozess wurde in Form einer geführten Tour umgesetzt.
Um dann noch die Effektivität dieses Lernszenarios in VR zu ermitteln, müssen alle Aktionen vom Lernenden in xAPI erfasst und im Learning Locker untersucht werden. Diese Abschlussarbeit liefert Hintergrundinformationen und Designentscheidungen für die Entwicklung einer VR Lernapplikation und die damit verbundenen Systeme für Learning Analytics. Damit umschließt die Arbeit die Erweiterung ebenfalls die Erweiterung einer Datensammellösung für Virtual Reality und die Entwicklung eines Tools für die Generierung von C# Klassen, um sicher zu stellen, dass die xAPI Komposition in C# konsistent ist. Des Weiteren wurde, im Umfang dieser Arbeit, eine Pilotstudie ausgeführt, um erste Schwierigkeiten der Implementierung oder Nutzung zu identifizieren und um erste Erkenntnisse über den didaktischen Mehrwert dieses Prototyps zu erlangen.

Chapter 1 Introduction

This chapter at first provide the motivation behind the use of learning technologies in VR and transition to the concrete motivation for applying this technology in a specific context *Rendering Pipeline in Computer Graphics*.

1.1 Motivation

Rapid technology changes affect education

Today's rapid changes induce new challenges for education, as Learners have to be prepared for continually changing society and economy [2]. But continuous increasing technology possibilities bring new way of learning. Technologies like YouTube, Social Media platforms, Learning Management Systems like Moodle enable high connectivity between teachers and learners and support fast exchange of diverse learning materials (graphics, videos, software etc.). These changes also affect higher education. Using technology in higher education can be cost effective, allows flexibility of time and place, enhances efficacy of knowledge, supports communication platforms, considers individual knowledge levels and thus allows learners to learn at their own pace [3]. Moreover, according to 2020 EDUCASE Horizon Report [4] the trends for higher education are manifold and can be divided into five categories: *social, technological, economic, higher education* and *political*. They cover many challenges like mental health, demographic changes, equity, Artificial Intelligence, analytics, privacy, climate change, change in student population, online education, political polarization and others. Further, they profile key trends and shape the future of teaching and learning. They sketch four possible scenarios (growth, constraint, collapse or transformation) for the future models of education for emerging technologies. Beside of established technologies like *MOOCs, flipped classrooms, mobile learning* and *makerspaces*, additional technologies and practices will have great impact on education. In general, virtual learning has several levels of interaction and immersion, beginning from real-life simulations over to construction safety or medicine and health education [5]. One collection of the emerged technologies are XR (AR, VR, MR, Haptic) technologies. They are more immersive technologies, which extend the reality and let the learner be part of the experience. XR generally shows great potential for learning and teaching. It can effectively support skill-based and competency pedagogy [4]. Immersive learning environments make classes more interesting. Learners can acquire the matter easier, pay better attention on learning material, raise interest for selected topics and furthermore memorize the acquired knowledge longer [6], [7]. Also case studies in [8] showed positive influence of VR's features on training of problem-solving capability, motivation and learning performance. Moreover, VR shares detected advantages of CAI (Computer Assisted Instruction) [5]. For example, because of VR's possibility for free exploration and immediate feedback, it can grab and hold students' engagement and encourage them to be active learners. Especially constructivist approaches let students create new experiences and learn by doing. Looking onto the current and upcoming world's global

cost effective, time/place flexible, enhance efficacy of knowledge, individual knowledge levels

VR can have possitive effects on education

Rethinking educational models is more important then never before

challenges (e.g. climate change, increasing student population), the transformation scenario explained in [4], makes it urgent to rethink traditional learning systems now more then never before. In addition of already known, this year comes up with a problem no one had actively considered before: the Corona pandemic. This crisis has indeed challenged the regular education systems (schools, universities, etc.). Distance learning tools become more important [9] and to prevent more learning loss they are needed to follow students' attendance, behavior and learning progress and provide individual support for the improvised "new" models of education [10]. No one knows how long this crisis will be part of everybody's life. Thus, not only for this pandemic, but also for possible future crises, it is necessary to upgrade our ways of teaching. Virtual learning technologies themselves cannot resolve education issues during a crisis, but they can help to organize classrooms and deliver learning materials. Their coverage for M. David Merrill's five principles of learning [5] (section 2.1.1) and their potential to facilitate distance learning and enable individual student support, make them good facilitators of today's and future problems. Already 2017 investments in several virtual technologies (including HMD-based VR) have been increased [5]. It is still not clear how to exactly integrate such technologies into the educational process in a solid way. Thus, it is important to understand effective instructional designs [11]. To achieve this, it is necessary to understand effectiveness of several technologies and the context they are applied on. This includes designing and implementing VR applications for educational purposes and evaluating their effectiveness. For this kind of VR application, it is possible to build immersive visualizations of complex topics and finally analyze user interactions and reactions to find out effective approaches.

1.2 Hypothesis

Is Learning in VR more effective?

Initial discussions for this thesis have lead to the thoughts, that the whole visualization may not be an effort for students with pre-knowledge. It may give good support for younger students (e.g. pupil). Thus, we started to ask *"Is it more effective to learn the Rendering Pipeline in VR as with native techniques?"* and *"Is learning via VR generally more effective?"*. VR offers, in opposite to native techniques, to be part of the experience via real-time visualizations and interactions. Based on these thoughts, there were two major hypotheses created: *(1) "Knowledge which is achieved via experiments or visualization holds longer in long-term memory."*and *(2) "Interaction is essential for learning via VR."*.

1.3 Thesis Goal

Evaluated pilot prototype

The Rendering Pipeline (RP) is a fundamental part of *Computer Graphics (CG)*. It starts with transformation of three-dimensional models and ends in displaying two-dimensional images on the user's screen. This whole process consists of several stages (*Application, Geometry* and *Rasterizer*) with possible sub-stages in each stage [1]. The aim of this thesis will be to analyze this concrete learning scenario, create a concept for it and develop a first pilot prototype of a VR learning application in Unity, where a user can observe the Rendering Pipeline in VR. In this environment it will be possible to interact with objects of the scene to get an idea about the functionality of a Rendering Pipeline. Additionally, a pilot study is planned to identify first issues and to evaluate results for further iterations of a design-based research cycle to investigate the hypotheses.

Chapter 2 State of the art

This chapter will at first dive into scientific background of *Technology-Enhanced Learning* (TEL), *Virtual Reality* (VR), *Computer Graphics* (CG) and the *Rendering Pipeline* RP. Further it will present related work in research and market to discover their Virtual Educational Environment (VEE) type and virtual world (VW) class.

2.1 Scientific Background

Edu. VR VR app educational purpose

The usage of VR in education has increased the last years [12]. One concept of many others are Educational VR Applications. As this work will focus on teaching the Computer Science topic: Computer Graphics, it has an educational purpose. This makes the implemented prototype of this thesis to an Educational VR Application [12]. Thus, it is necessary to consider didactics background and compare existing educational systems.

2.1.1 Technology-Enhanced Learning

TEL aims to improve quality of learning

The benefits and downsides may differ by application field

Clear definition of goals and limitation is needed

Technology-Enhanced Learning (TEL) can be referred to as "integrating the use of digital technology into the learning and teaching process to improve the quality of learning" [13] and subsumes the older term *e-Learning* [14]. But as there are many definitions of e-Learning, it is still important to identify a common definition [3]. Beside of different types of e-Learning, the ways of employing them in education result in the types *adjunct, blended e-Learning* and *online* [15]. Therefore, Arkoful and Abaidoo [3] checked the adoption of e-Learning in education by comparing different literature and summarized some advantages and disadvantages. E-Learning can increase the flexibility of choosing the time and place of learning, enhance the efficacy of knowledge, access huge amount of information, connect learners, be cost effective (e.g. reducing traveling), consider learners differences, help compensate for scarcities of academic staff and provide self-pacing. But the downsides can be lack of relations or interactions, negative effects on communication skills, danger of cheating on tests, piracy, plagiarism, decreasing of the socialization role of institutions, increasing technology deployment costs and is possibly not applicable on all fields. Even though there is a considerable growth in the usage of technology in higher education and it can have great benefits, "there is still much to be learned about the effective educational contribution" [14]. Kirkwood and Price mentioned in [14] that the term TEL is too often used in an unconsidered manner. The design of evaluation and result interpretation require clear definition of goals in particular educational context. They further recommend the recognition of limitations and state those. Thus, the following paragraphs explain important aspects of TEL's side of this work to build later a connection between TEL and Virtual Reality.

Didactics and Learning Theory

Learning theory models learning progress

Learning itself is a teacher's witting strain to support students that they can learn new skills or information. But another interpretation of this term can be done by *learning theories*. A learning theory can be understood as a description attempt, how humans learn. They were constructed to discuss and understand how learning is happening. Prevalent learning theories are *behaviorism*, *cognitivism* and *constructivism* [16], [17]. Briefly: behaviorism focuses on conditioning and cognitivism on cognitive insights. The central idea of constructivism is that learners construct new knowledge upon foundation of previous knowledge. Further learning theories are *humanism* [16], [17] and *connectivism* [17], [18]. Humanism focuses on the specific human capabilities (personal growth, choice and creativity), whereas connectivism is more sensible in online learning scenarios [19], because of its role of social and cultural context and learning arises here by creating networks.

behaviourism, cognitivism, constructivism, humanism, connectivism

According to M. David Merrill, the most effective learning progresses (or environments) are problem-centered [20]. Learning is promoted if (1) new knowledge is demonstrated to the learner, (2) learner applies the knowledge, (3) old knowledge is activated as foundation for new knowledge, (4) new knowledge is integrated in learner's world and (5) knowledge can be assigned to meaningful tasks [21]. A constructivism-based approach turned out to be very effective [8], because of its high degree of interaction. In fact, this fits to VR (more later). Anyway, real learning effectiveness of a VR application shall be explored by analyzing the virtual environment by applying, for example Learning Analytics.

Learning Analytics

On the LAK11 (Learning Analytics and Knowledge) website[1], Siemens defined Learning Analytics (LA) as "the measurement, collection, analysis and reporting of data about learners and their contexts, for purposes of understanding and optimizing learning and the environments in which it occurs". Chatti et al. [22] characterized Learning Analytics into a cycle of three major steps: (1) data collection and pre-processing, (2) analytics and action and (3) post-processing. An illustration of this cycle can be found in figure 2.1. Aside from that, they described a *"A Reference Model for Learning Analytics"* with the four dimensions (*What?*, *Who?*, *Why?* and *How?*). This model helps to clarify a context and to specify this way *Data and Environment*, *Stakeholders*, *Objectives* and *Methods* for LA. Analytics results could help in future decisions or to identify issues in current realizations.

LA as cyclic process
LA Reference Model: What?, Who?, Why?, How?

In this work, the stakeholders (Who?) are researchers, learners (pupil and students), developers and teachers (see section 3.2.1). Analytics results are used to assess the learning achievements and to identify learning outcomes of presented VR scene as learning material (Why?). Learners will have first-person experiences, take decisions and interact after obtaining feedback [23]. To reflect a learner's pathway, it is effective to collect learner's "breadcrumbs" [23] (What?). Those can consist of objects the learner focuses or interacts with, visited locations, head rotations or other kind of interactions. To express the breadcrumbs, it is necessary to use a kind of language (How?), that is able to create sensible log statements. For this purpose, there already are technological standards to create statements, which can be used to describe learner's activities. They will be discussed later in chapter 4.3.

Collect activity breadcrumbs to reflect learner's pathway

[1] LAK11: `https://tekri.athabascau.ca/analytics/` (visited on 08/17/2020)

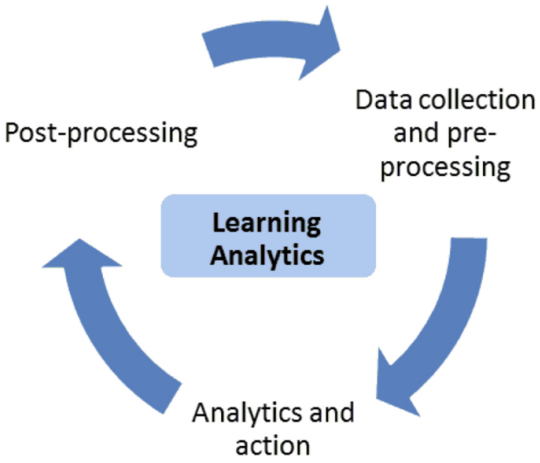


Figure 2.1: Learning Analytics Process [22]

Learning Analytics Data Collection

Usage of xAPI and Learning Locker for Data Collection

Learner activities can be logged. One way to give those logs a structure is to have a kind of grammar. Two established standards for this in context of Learning Analytics and Educational Data Mining are xAPI and IMS Calipher. Ehlenz et al. have systematically treated xAPI in [24] and decided to work in their research (Learning Technologies Research Group) with it. For storing such standards in a suitable manner, a Learning Record Store (LRS) is needed. Lukarov et al. compared critical criteria for several LRS implementations in [25] and finally decided to use Learning Locker. As it makes sense to adopt these decisions for my work, there will be no more comparisons for such LA technologies. More information about both used technologies can be taken from section 4.3.

Learning Technologies

VR as an effective learning technology

Not only is Learning Analytics an essential part of TEL, technology itself is it too. Apart from presentation forms like *Prezi*[2] or Learning Management Systems like *Moodle*[3] [26], there are more interactive technologies like *TABULA*[4] and Virtual Reality. From the perspective of learning technologies, VR has positive research results about the usage for teaching mathematics and geometry ([27]) and its effectiveness in K-12 and higher education [7], [28]. This makes it interesting to be considered more from a perspective of learning technologies and to that effect of TEL.

2.1.2 Virtual Reality

According to Confucius' quote "Tell me and I forget, show me and I may remember, let me take part and I understand", it could have a positive effect to integrate the learner into the experience. This wisdom is supported by data that show people can remember 20% of what they heard, 30% of what they saw and 90% of their personal experience or simulation [29]. One possible technology to achieve personal experience inside a simulation is *Virtual Reality* (VR) [30]. The concept refers to a whole simulated reality the user can dive deep into and interact

[2] Prezi: `https://prezi.com/` (visited on 10/30/2020)
[3] Moodle: `https://moodle.de/` (visited on 10/30/2020)
[4] TABULA: `https://learntech.rwth-aachen.de/go/id/ltmn` (visited on 10/30/2020)

Immersion, Interaction, Imagination are important for LA

with via VR devices (e.g. 6DOF: 6-Degree-of-Freedom Controller) [5]. The term "Virtual Reality" is defined as I^3 for "Immersion-Interaction-Imagination" [31].
Immersion can be understood as *the illusion of being there* [5]. It can be classified into mental and physical (or sensory) immersion. The sensory immersion creates user's illusion via sensory information (like realistic graphics, environmental sound effects, etc.). Mental immersion gets its strength by content's design (e.g. a good story can also create an illusion of being inside the story).
According to Huang et al. [8], interaction plays a major role in VR, because users do not only want to see, but also manipulate the virtual object. Depending on equipment and system, the user is also able to touch and feel the object with his/her physical sensors (visual, auditory, haptic, tactile, smell and taste). But VR is not only a playground for interaction, it is also interesting for real-world problems. Users want to address issues that require imagination and creativity to solve them (e.g. engineering, medicine, space exploration, biology) [8].

Robust strategy of learning content is important

Imagination helps the learners to visualize new ideas and develop solutions for problems, where creativity is needed [8].
Even though all three VR features (I^3) are relevant indicators for Learning Analytics, educators don't have to deploy all of them when delivering a Virtual Reality Learning Environment (VRLE) [8]. Effective learning outcomes are not dependent only on the technology itself. Robust strategy and delivery of didactics concepts play a stronger role in learning environments [32]. Vergara et al. proposed a design to improve the level of meaningful learning in a VRLE in [33]. It has many steps, but those can be grouped into: determination of VR usefulness, technical design, development and evaluation. Thus, the following sub-sections will prepare immersion, interaction and imagination nature theoretically from didactic manner and afterwards followed chapters will fulfill the proposed VR design.
In the following, after the classification of immersion comes the classification of interaction and finally a brief part of possible limitations.

Types of Virtual Educational Environments

Degree of realism shall depend on learning content

To better understand criteria for immersion of a VRLE, it is necessary to examine several types of VR environments. A system can be classified into immersive (e.g. HMD's, CAVE[5]), semi-immersive (e.g. mobile AR) and non-immersive (e.g. computer displays) [34], with each of them having different degree of realism.
To couple the degree of realisms onto educational context, a taxonomy of *Virtual Educational Environments (VEE)* by Kamińska et al. [12] may help out. It strictly connects the taxonomy **(1)** remembering and understanding, **(2)** using acquired knowledge in typical situation and **(3)** using acquired knowledge in a challenging situation to the levels of immersion. An own illustration can be seen in figure 2.2. In detail: the first type is mainly used to state knowledge of a particular field of science (dates, facts, rules or scientific theories). It has the lowest level immersion and can be realized with simple hardware like mouse, keyboard and screen. It is effective for example in history education [35]. The second type is used to teach practical skills. In such semi-immersive scenarios the learning experience consists of earning theoretical knowledge from a presentation and afterwards practice in an immersive environment. This can be useful, for example, for teaching science or social studies [27]. The third-typed environment would challenge the learner directly inside immersive environment and face the learner with

[5] Cave Automatic Virtual Environment

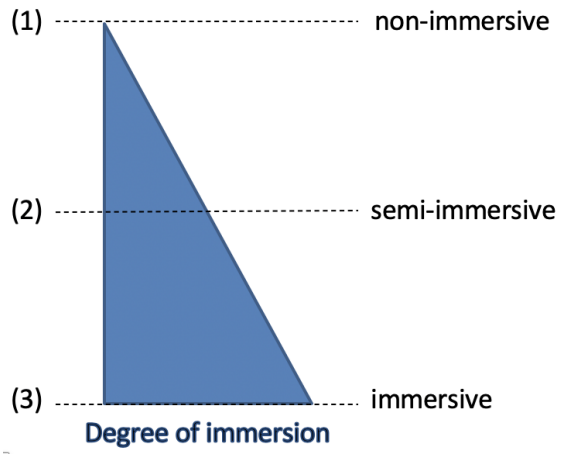


Figure 2.2: Relation between Virtual Educational Environment types to degree of realism. Legend: (1) remembering and understanding, (2) using acquired knowledge in typical situation and (3) using acquired knowledge in a challenging situation

problems, they have to learn how to solve. This scenario is mostly interesting for example in medicine or engineering [12].

Classification of VR worlds

Degree of interaction depends on virtual world design

As every *Virtual World* (VW) has another degree of interactivity [36] design of VR interactions requires considering different types of virtual worlds in VR. According to Schwan and Buder [37], there are *exploration*, *training*, *experimental* and *construction world*. An illustration of the relation between the degree of interaction and classes can be seen in figure 2.3. The first one allows free exploration at own pace and order, because of constructivism and cognitive flexibility. The second one (training) is for behavioristic-cognitivistic scenarios, which need to be repeated and trained. Thirdly, experimental worlds visualize causal relationships and shows the user changes, when he/she manipulates parameters (also called discovery learning). And construction worlds give the opportunity to directly create own virtual objects or environments. This type requires often pre-knowledge and supports the principle of constructivism.
Weise and Zender compared these world classes with some applications and classified each application to multiple classes [36]. This let them define an additional class called *exposition*. An exposition world guides the learner through the scene with a fixed path and sequence of learning content. This world has a reduced degree of interaction, but provides guided learning. While immersion and interaction can be easily classified, imagination doesn't follow strict design and depends on learning content. Thus, the next sections provide ground for design decisions.

VR in education

VR could complement teaching

In education, VR could help to improve the way of teaching and learning. According to Dong [29] VR can support traditional schools and do not replace them. VR classrooms, for example, can be fully compatible with traditional school classrooms . A VRLE usually simulates a classroom or laboratory [12]. This also makes it possible to simulate harmful or dangerous experiments while staying safe [29].

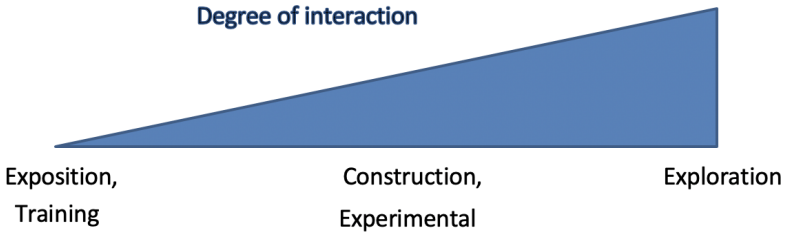


Figure 2.3: Relation between Virtual Reality worlds classes to degree of interaction

But beside of previously mentioned potential positive influence of VR in education, Stojšić et al. [6] united some lacks for this technology for using VR in teaching:

There are still not-ignoreable lacks

- High equipment costs make this technology less affordable.
- Internet connection problems in classrooms would disturb teaching in VR.
- There should exists easy-to-use authoring tools for creating VR experiences.
- Increased time for class preparation.
- A main obstacle is the lack of familiarization and motivation of using VR technologies in teaching by many teachers.

Great hope in cost reduction and 5G

There is great hope in new lost cost HMD, WebVR technologies or tools like *Google Tour Creator* [6]. WebVR could be, for example, optimized for remote classroom participation [38]. The power of 5G technology may solve the future Internet connection problems and support wireless HMD's [39], [40] (reduce setup complexity). Already these days, the effectiveness for constructivism, simulation-based and active learning [8], [41] promises good support for training scenarios. The commitments were checked by testing fields like mathematics, biology, physics, chemistry, astronomy, engineering education and many more [29]. But not only constructivism based scenarios could benefit from research of VR technology for education. Those scenarios, where learners repeatedly practice (behaviorism) new skills in safe environment, can also benefit. For example, prospective teachers can be prepared for real education in a virtual classroom [42], while having the chance to correct failures and repeat estimated actions.

Today realistic for job preparations

Motion sickness, Cybersickness, VR sickness

VR sickness shall be always considered for design

Apart from previous mentioned lacks, VR itself has some issues. Motion sickness [43] (more generic VR sickness [44] or cybersickness [41]) is one of them. It describes physical discomfort of the VR user like eye strain, headache, pallor, sweating, nausea, vomiting, disorientation etc. [45]. As there are several theories for motion sickness (*Sensory Conflict Theory*, *Evolutionary Theory*, *Postural Instability Theory*, *Rest Frame Hypothesis* and *Eye Movement Theory* [43]) and it is not clear, which one is correct, this thesis will not go deeper into them. The major causes for this issue are hardware, content and human factors [44]. Thus, these factors have to be considered for VR system design decisions and will be discussed in evaluation if necessary.

VR Tour

Exploration guidance by an "tour host"

According to Velho et al. [46], a tour can have four types: (1) *Meta-Narrative*: content is associated with a story, (2) *Narrative-Based Content*: story with narrative structure (e.g. theater play, documentary), (3) *Navigation*: exploration via navigation through the environment, (4) *Guided Tour*: exploration is guided by a "host" and (5) *Participatory VR*: guests experience the tour in a shared VR environment via interaction. In [46] a VR tour structure generally consists of main logic parts (sequences), parts of the story (scenes) and sub-parts of scenes for meta-narrative texts or clips (segments). The content can be composed by text, story clips and non-linear dependencies between different parts of meta-narratives.

2.1.3 Computer Graphics

CG is the useage of computer to create images

The term *Computer Graphics* (CG) describes the usage of computers to create, manipulate and display images on screen. The work with Computer Graphics needs knowledge of hardware, file formats, algorithms and using a graphics API (like OpenGL or DirectX) [47]. This knowledge can be used to create images from a virtual scene. In Computer Graphics' context a virtual scene consists of three-dimensional (3D) models. A *model* defines a collection of geometric entities, where each of them consists of geometric primitives (mainly triangles). These primitives can be spanned by *vertices* (singular *vertex*), which are defined by vectors in a three-dimensional space (in this work $\mathbb{R}^3$) describing 3D positions and colors [1]. The translation of 3D geometry into two-dimensional images will be done by efficient 3D graphics *Rendering Pipelines* (RP's) that modern computers have [47].

RP is a sub-system to project 3D primitives on screen

Rendering Pipeline

RP concept: three main stages, many sub-stages

A RP is a special sub-system to draw 3D primitives in specific perspectives [47]. Thus, the Rendering Pipeline is a fundamental part of Computer Graphics. This whole process consists of several stages (*Application*, *Geometry* and *Rasterizer*) with possible sub-stages in each stage. A basic model of Rendering Pipeline can be seen in figure 2.4. The following details are taken from literature "Real-Time Rendering" by Akenine-Möller et al. [1]

scene, physics, collisions, inputs

Application Stage The application stage is fully controllable by the developer and every operation is executed by the CPU. Here, the developer defines the geometry (rendering primitives like points, lines and triangles), which get provided to the Geometry Stage. Further, the developer implements physics and collision detection here. Handling of user device inputs, such as keyboard or mouse, are also realized here.

model transform: translation, rotation and scale

Geometry Stage The Geometry Stage runs per-polygon and per-vertex operations. It is divided into the functional sub-stages: *model and view transform*, *vertex shading*, *projection*, *clipping* and *screen mapping*. As a model is a collection of vertices and triangles, each vertex is defined by a local position relative to the model's origin inside the model (*local space*). Those have to be transformed into a space with one common origin (*world space*) to check, for example, whether one 3D object is occluding or intersecting another. This transformation is done by multiplying the model's world information *translation matrix* (position), *rotation matrix* and *scale matrix* into one matrix (*model matrix* or *TRS matrix*). This step is called *model transform*. To bring all world's objects into perspective of the user, they have to be regarded from user's position and perspective. This can be solved by placing a virtual camera into the

virtual camera = user's perspective

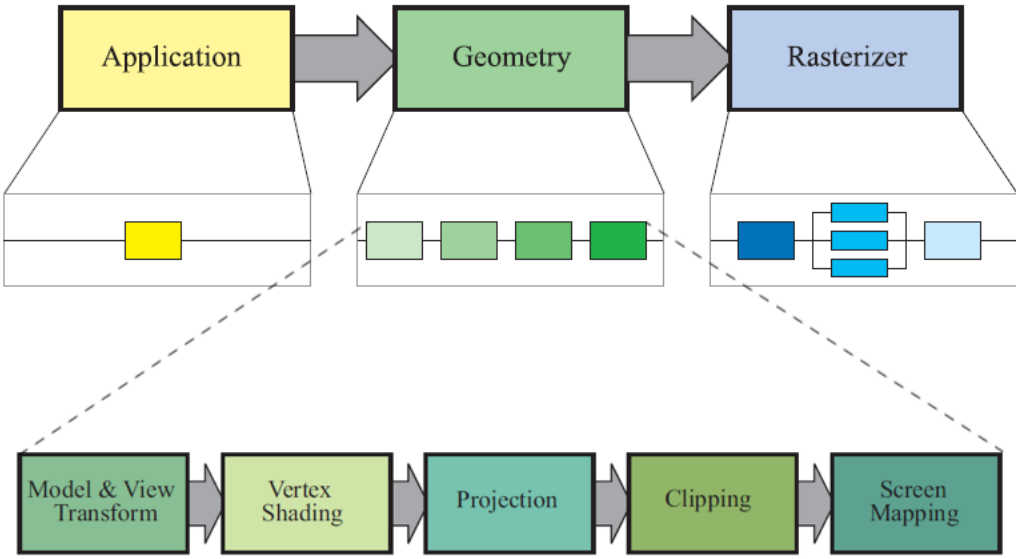


Figure 2.4: Basic model of a Rendering Pipeline on the top and detailed view of Geometry Stage in the bottom. Combined images from [1].

*TRS = Translation * Rotation * Scale*

scene, which represents user's view perspective. As the camera also has a location and rotation in world space, it also has a TRS matrix. To converge user's perspective, all scene objects have to be transformed into user's view (*view space* or *eye space*) by being multiplied with the inverse of camera's TRS matrix. This step is called *view transform*. Now, the camera is looking from the origin at negative z-axis, having y-axis upwards and x-axis pointing to the right. A truncated pyramid exactly defines the camera's view space and is called *frustum* (or *view frustum*). In figure 0.1 the camera's origin is painted as a dot in the top right corner. Lines on the sketch form frustum's shape to illustrate camera's view space. Technically it is done by defining the frustum's plane positions and angles for field of view horizontally and vertically. In the projection stage, this frustum-based view volume is transformed into a unit cube (*canonical view volume*) with $(-1,-1,-1)$ and $(1,1,1)$ as extreme points and the target rendering scene is normalized into *normalized device coordinates*. As a result, the scene is deformed to create the perception of different objects' sizes. Literally an object far away from user, shrinks more and visually it gets smaller (right side of figure 2.5). This kind of projection is called *perspective projection*. Another option is the *parallel (or orthographic) projection*. Parallel lines in the picture will stay parallel here (left side of figure 2.5), because the camera has a rectangular box instead of a frustum shaped view space.

Frustum defines user's view area

Projection: transform scene into user's perspective

To reduce computation time, only from user's perspective visible primitives will be rendered. Those primitives which are partially inside and outside the view volume will be discarded by *clipping*. In this stage, primitives are clipped along the canonical view volume. Unlike the previous stage, clipping is done by fixed hardware operation.

Clipping reduces render complexity

The last sub-stage is *screen mapping*. The x- and y-coordinates of clipped primitives are transformed into *screen coordinates*. Together with the z-component (with $-1 \leq z \leq 1$) they are called *window coordinates*. These coordinates are passed to *Rasterizer Stage*.

Rasterization is the detection of pixel colors ⇒fragments

Rasterizer Stage The *Rasterizer Stage* is divided into the functional stages: *triangle setup, triangle traversal, pixel shading* and *merging*. More technical details about this progress are not relevant for this work and can be taken from [1]. In summary, the rasterizer computes and sets

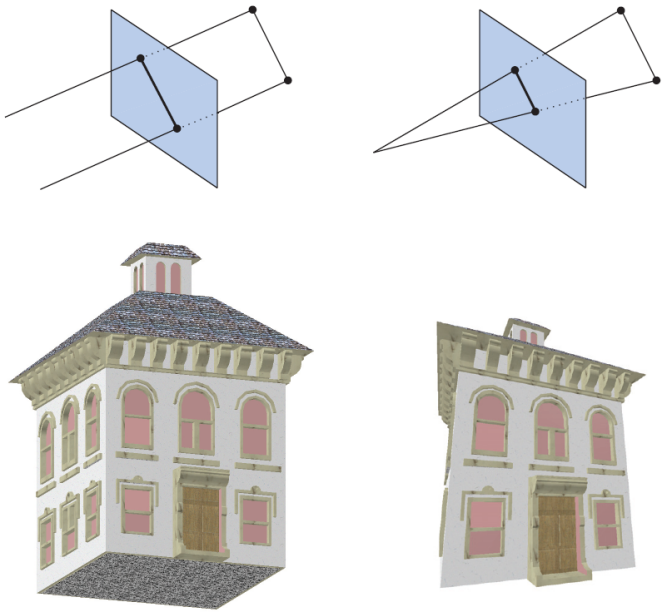


Figure 2.5: Orthographic projection on the left and perspective projection on the right [1]

colors for screen pixels. The *rasterization* (or *scan conversion*) creates *fragments* (potential pixels) by detecting visible primitives of the scene. While testing visibility of all fragments a, so called, *z-buffer* (or *depth buffer*) is used to store the actual nearest z-value of tested fragments and in the end the *color buffer* is used to pick correct mix of colors for the pixels.

This model is conceptual and complex

But this model of the Rendering Pipeline is very conceptual and too complex in detail for a first prototype. The idea of Rendering Pipeline can be well described by simpler models.

Simplified Rendering Pipeline Model

A simplified model can be used to describe RP

The Computer Graphics Group at RWTH Aachen University offers a lecture called Basic Techniques in Computer Graphics by Prof. Dr. Leif Kobbelt (Visual Computing Institute). The Rendering Pipeline is described there via a simplified model (see figure 2.6): *3D Geometry* represents a scene's 3D geometry and the previously described *Geometry Stage* is divided here into *3D Transformation/Perspective Transformation* (summarized for *Model & View Transformation*, *Vertex-Shading* and *Projection*), *Clipping* is analog to the previously described clipping mechanism and the *Rasterizer Stage* is described here by the four simples stages *Rasterization*, *Lighting* computations, object *Texturing* and fragment's *Visibility Test*. Then the last one outputs the *Image* as result of the pipeline.

Shaders

GPU programs for rendering customizations

Shaders are a discipline of CG and this term will be needed later. Thus, a brief understanding will be described here. The term *Shading* generally means the computation of using an equation of radiance (measure of brightness and color of a single light ray) along the view ray based on light source and material properties [1]. Shaders are programmable parts of the Rendering

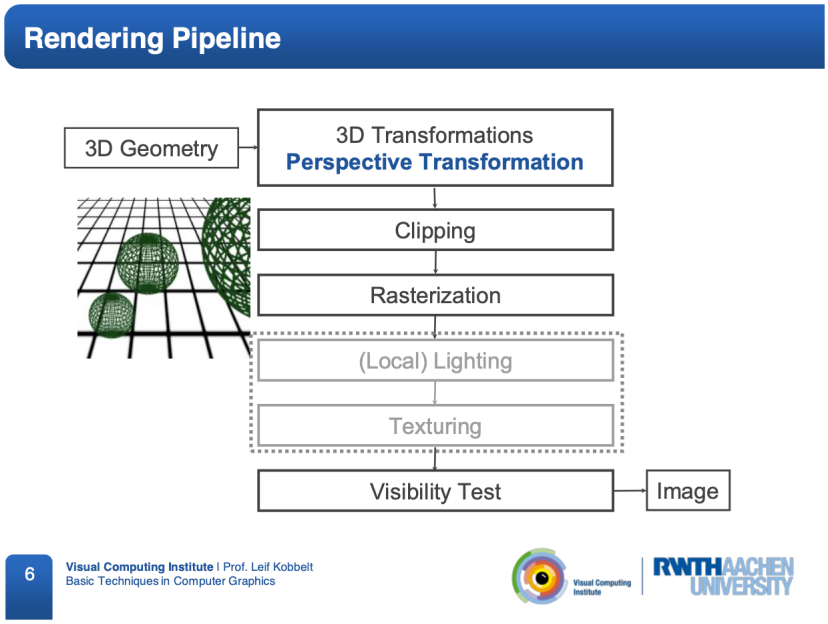


Figure 2.6: Simplified Rendering Pipeline Model. Retrieved from CG Lecture 04 Slide 6, Prof. Dr. Leif Kobbelt [48]

define behavior for vertices and fragments

Pipeline [49]. A developer can use a shader language (e.g. GLSL[6] for OpenGL or HLSL[7] for DirectX) to write these kinds of programs for the RP. As the shader program is executed by GPU, graphical operations are done efficiently. Two well known shaders are vertex and fragment shader. In both, cases a shader program is only describing the behavior of one vertex (or one fragment) and this behavior will be applied on all elements. For example: the vertex shader scales the given vertex by factor two and returns color red in fragment shader. The whole mesh will be painted in red color and is twice bigger than originally.

2.2 Designing an Educational VR Application

iterative design and design-based research are cyclic

Reeves asserted in [50], that there are some problems with instructional technology research and poor quality of educational technology research. Hence, he compared some applied research methods on education research and listed some heuristics for instructional technologies. Those are covered by a "new" research method: *design-based research*.
Especially VR design differs from classical 2D desktop or mobile application design. According to [43], a VR design must be iterative and be based on continual redesign, prototyping and feedback from real users (*iterative design*). It further needs thoughts about used hardware, applied selection and travel techniques.

2.2.1 Design-based Research

long-term findings

A research method for education research is *design-based research*. It is designed by and for educators to possibly increase the impact, translation and transfer educational research results into improved practice [51]. According to Shah et al. [52], design-based research studies

[6] GLSL: `https://learnopengl.com/Getting-started/Shaders` (visited on 10/30/2020)
[7] HLSL: `https://docs.microsoft.com/en-us/windows/win32/direct3dhlsl/dx-graphics-hlsl-pguide` (visited on 10/30/2020)

foresee to get long-term findings through multiple iterative cycles with a single larger design-based research study. While a macro-cycle of those studies focus on theoretical fundamental understanding and knowledge generation, smaller cycles (the micro- and meso-cycles) target to generate practical knowledge, based on use and implementation of the vision. Overall the design progress can be abstracted into the cycle (see figure 2.7) of the steps *analysis*, *development*, *evaluation* and *reflection* [53]. Tested results can help to refine problems, solutions and methods and therefore continuous improvements of an implementation can increase the learning experience of an approach.

analysis, development, evaluation, reflection in cycle

2.2.2 Iterative Design

early prototype, continous feedback

Simple VR experiences can be built in a short time period and feedback should lead to redesign to improve the experiences [43]. Requirements have to be discovered in progress. Thus, it is important to collect feedback as quickly and as often as possible from stakeholders and experts. At start it is important to begin implementation as early as possible, and in the end there is more value in having a "bad" prototype working instead of "overthinking" the design and spending too much time in considering and debating each factor (continous discovery principle). At all times, the development should focus on the user experience and give up traditional ways of software measurements (e.g. lines of code metric). According to Jerald [43], a good way is to apply *iterative design* by a *Define-Make-Learn Cycle* (figure 2.8). It can help to optimize user experience and make VR more comfortable. Even though the stages are conceptually performed in sequence, they often occur in parallel. It tightly depends on project specifications. All three stages are explained in the following. Jerald has deployed many guidelines for a lot design challenges in his book [43]. Because of limited time capacities not every guideline will be applied in detail.

define, make, learn cycle

Define Stage This stage tries to define what is planned to make. It includes everything from high-level vision to listing requirements.

Make Stage The *Make Stage* is for clarification of "how to realize" the vision.

Learn Stage In this stage it is important to reflect the realization and see what works and what does not work. The results are used to get back into *Define Stage* and refine what is made.

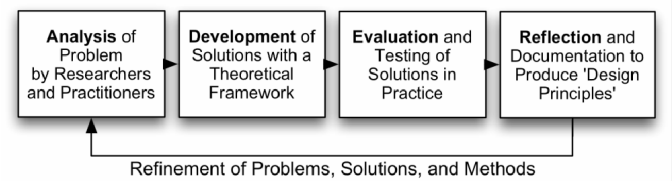


Figure 2.7: The design research cycle (Adapted from [50]). Taken from [53].

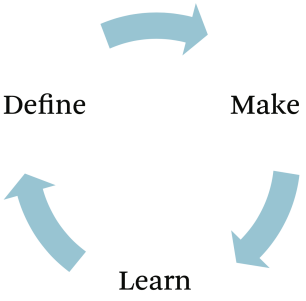


Figure 2.8: High-level iterative process for VR development [43]

2.2.3 Metaphors and Techniques in VR

Interaction techniques is part of decision

According to Jerald [43], design decisions in VR have to think about 3D user interactions. This includes thinking about *Human-Centered Interaction* (e.g. intuitiveness), *VR interaction concepts, input devices, interaction patterns (or metaphors) and techniques* and *viewpoint control patterns (travel/locomotion)*.
Interaction metaphors are mental models of a technique and techniques are concrete realizations of a metaphor. To denote the beginning of a manipulation task, there needs to be a specification of a set of objects. This specification is called *selection* [54]. Selection of an object can be done via metaphors pointing, touching or saying its name and is realized by using tracked instrumented gloves, joysticks, VR controllers or other devices. More details about the chosen device can be read in section 4.1.

Selection patterns

pointing, hand selection, image-plane selection and volume-based selection

A selection task in VR is more complex than on desktop. There are several selection patterns: *pointing, hand selection, image-plane selection* and *volume-based selection*. Each of them has advantages and limitations ([43], chapter 28). A brief comparison of the limits and usage reasons can be seen in table B.1. Generally interaction itself is a big task and can be subcategorized into smaller techniques like *grasping*, several pointer approaches or image-plane selection related approaches.

Locomotion patterns

Locomotion technique depends on app criteria

Traveling through virtual space may influence the immersion. Nowadays, it is impossible to walk through infinite space in a small room. Thus, there are many different approaches in direction of *physical locomotion* (like redirected walking [55]).
In contrast stand "pragmatic approaches". The two main categories here are *steering* (directed by gaze, hand, torso and lean) and *pointing* (target-based or route planning) based approaches. Each locomotion has to be chosen depending on application criteria.

2.3 Related Work

As my work aims to realize an Educational VR Application prototype, another VR applications for educational purposes could contain usable findings. In the following are firstly some scientifically evaluated approaches, afterwards some of the newest systems on the market. Finally, I assigned and compared the setup, VEE type and VW classes of related approaches in table 2.1.

2.3.1 Scientific Approaches

Scientific approaches are mostly evaluated approaches with transparent upsides and downsides. In the following four scientific approaches are described.

Astronomy Teaching

visualization of solar system, free navigation

Yair et al. created a Virtual Reality application for Science Education and present the results in "3D-Virtual Reality in Science Education: An Implication for Astronomy Teaching" [56] and used it later for a thinking journey concept [57]. They focused on the support of a scientific visualization for the solar system. The user can navigate through space and explore information about planets by touching them. They distinguished between four different view modes a teacher could (for example) instruct to switch: Free-Mode, Sun-in-Site view, Planetary view and Geocentric view. They concluded powerful learning experience and enhanced the awareness for planet distances and activities.

Construct3D

geometric construction tool

Kaufmann et al. [30], [58]–[60] developed a geometric construction tool based on the collaborative augmented reality system "Studierstube" and called it *Construct3D*[8]. They've used HMD and a Personal Interaction Panel for 3D interaction. In their study, audio assistance by invalid inputs brought positive results. They've designed several tasks to learn the interaction and "First-Time-VR-Students" took longer than planned. They concluded that VR is a good playground for experiments and students had walked around their creation to be proud of what they made. The possibility to create some things lies in nature of this project, but this observation highlights the importance of constructivism in a Virtual Reality Learning Environment.

CyberMath

CAVE, multi-user collaboration, guidance

CyberMath [61] is an interesting approach from the year 2001. It uses *DIVE* [62] (a multi-user Virtual Reality system for CAVE and HMD environments) on the ground. Because most VR-based systems focus on constructivist learning theory, they let students collaborate with each other, but believed that teacher's guidance has positive effects. They initiated a Guided Tour to get a sense of complex shapes. Their two usability tests resulted that it is possible to successfully explain complex mathematical concepts in their system (a VR system).

IVR Museum of Color

color theory, virtual museum, increasing difficulty

IVR stands for *Immersive Virtual Reality*. Spalter and her team created a CAVE-based IVR application [63] to teach highly abstract aspects of color theory through a series of hands-on experiments. Every experiment takes advantage of immersion space and lets users interact with geometric explanations. They decided to design a virtual museum (architectural metaphor) to guide the learner through learning sequences. Each floor has the same layout and flow sequence but has different experiments with increasing difficulty. They were confronted with a trade-off between metaphor fidelity and convenience. A recommendation is to take aware of IVR interactions like note- or record-taking, narrative structures, changing scenes and tracking user's mistakes.

2.3.2 Market Approaches

The pictures were taken from their websites (listed in footnotes) and put their pictures together in figure 2.9.

[8] Construct3D: `https://www.ims.tuwien.ac.at/projects/construct3d/` (visited on 24. July 2020)



Figure 2.9: Related market approaches from left to right: CalcFlow, InMind 2, Titans of Space, Tilt Brush and VRMath

live manipulation

CalcFlow

CalcFlow[9] lets the learner discover the mathematical concepts of vector calculus, parametrized functions and more. The user is able to manipulate parameters and observe the results.

mobile VR solution

InMind 2

InMind 2[10] is a free scientific VR Cardboard game, where the learner can observe the chemistry of human's emotions. This game delivers scientific details and lets the learner drive a fixed tour. While taking this tour the learner takes decisions by rotating their head and focusing on an action element for a few seconds.

Guided Tour through solar system

Titans of Space

Titans of Space PLUS[11] is a fully immersive Guided Tour through the solar system. It can be used with HMD-based VR system and 6DOF controllers (*Oculus Quest*[12], *Oculus Rift*[13], *HTC Vive*[14] and *Microsoft Mixed Reality* headsets). It also has a lighter version (*Titans of Space Classic*) and a mobile version for *Oculus Go*[15] and *Samsung Gear VR*[16]. As this application is also available for desktop screen, it offers different shapes of immersion and interaction.

paint in 3D with virtual paintbrushes

Tilt Brush

With Google's *Tilt Brush*[17] the user is able to paint in VR's 3D space with different virtual paintbrushes and special effects. It is also available for, already in "Titans of Space" mentioned, HMD-based VR systems. Open source libraries give the possibilities to extend art. A small study on students' evaluation about this tool can be found in [27]. The tested children used a lot of creativity, but erased a lot of painting because of unfamiliar controls.

self learning of 3D geometry via mobile VR

VRMath

VRMath[18] is an interactive educational mobile VR app to support the understanding of 3D geometry, graphs and vectors. It is part of the educational platform *VARP Edu*[19]. VRMath provides a Virtual Classroom, the possibility of self-learning and progress statistics.

[9] CalcFlow: `https://www.nanome.ai/calcflow/` (visited on 07/24/2020)
[10] InMind 2: `https://www.luden.io/inmind2/` (visited on 07/24/2020)
[11] Titans of Space: `http://www.titansofspacevr.com/titansofspace.html` (visited on 07/24/2020)
[12] Oculus Quest: `https://www.oculus.com/quest/` (visited on 07/24/2020)
[13] Oculus Rift: `https://www.oculus.com/rift/` (visited on 07/24/2020)
[14] HTC Vive: `https://www.vive.com/` (visited on 07/24/2020)
[15] Oculus Go: `https://www.oculus.com/go/` (visited on 07/24/2020)
[16] Samsung Gear VR: `https://www.samsung.com/de/wearables/gear-vr-r323/` (visited on 07/24/2020)
[17] Tilt Brush: `https://www.tiltbrush.com/` (visited on 07/24/2020)
[18] VR Math: `https://www.vrmath.co/` (visited on 07/24/2020)
[19] VARP Edu: `http://www.varpedu.com/` (visited on 07/24/2020)

2.3.3 Comparison of Related Approaches

The previously presented applications are listed in table 2.1. Their VEE types, Virtual World classes and their availability was compared with each other. Half of the approaches have free access. Apps with In-App-Purchases were marked with (✓), because the free access is indeed limited. Most of them use HMD and give the opportunity to use a mobile variant (e.g. VR Cardboard). As the VEE types are for general virtual environments (not only VR), the type In-Mind 2 and Titans of Space have per definition VEE type (1). However, the most popular VEE are immersive (VEE type 3) and the most popular VEE class is *exposition* (four matches). The second most popular is *experimental* (two matches). The other VEE's classes have each only one match. Thus, it could be sensible to let the user do guide through the learning material (exposition) and partly allow experiments (experimental).

Most popular VEE are immersive and the most popular VW class is exposition and the second most is experimental

Application	**Setup**	**VEE Type**	**VW class**	**Free access**
Astronomy Teaching	VR gloves and helmet	(3)	exploration	✗
Construct3D	AR HMD PIP	(2)	construction	✗
CyberMath	HMD, CAVE	(3)	exposition	✗
IVR Museum of Color	CAVE	(3)	exposition	✗
CalcFlow	HMD	(3)	experimental	✓
InMind 2	HMD, VR Cardboard	(1)	exposition	(✓)
Titans of Space	HMD, Desktop, Mobile Gear	(1)	exposition	(✓)
Tilt Brush	HMD	(3)	experimental	✗
VRMath	VR Cardboard, Mobile Gear	(2)	training	(✓)

Table 2.1: Comparison of related approaches. VEE Types legend: (1) remembering and understanding, (2) using acquired knowledge in typical situation and (3) using acquired knowledge in a challenging situation.

Chapter 3 Conceptual Approach

big DBR cycle, smaller iterative design cycles

This master thesis is the first iteration of a design-based research study to create a pilot prototype, based on which new iterations will be continued. Thus it is needed to start to analyze the problems (section 2.2.1), continue with development and evalation and end up in reflection. Further researches then have to restart this cycle to refine the , here concluded, requirements and the research questions about the hypotheses in section 1.2. In addition, smaller iterative design cycles (figure 2.8) will be executed to follow VR design guidelines by Jerald [43].
To identify the didactic challenges (problems), which have to be resolved by a suitable prototype, a concept may be helpful. This chapter delivers a concept of the VR application and design starts with the *Define Stage* (section 2.2.2). This stage includes an overall vision from a user's point of view (section 3.1), serves a concrete understanding of objectives and use cases (section 3.2) and subsequently storyboards and presents the funded objectives.

3.1 Overall Vision and Objective

An overall vision represents the final potential of an application. Oriented on the overall visions, smaller visions can be created, which can be discussed more concretely and pragmatically. In consequence, goals can be formalized without loosing the overall vision. The following visions were discussed with the supervisors.

3.1.1 Visions

learn stages of the RP

As an overall vision the user should be able to impressively learn the stages of the Rendering Pipeline. The application provides texts and visualizations to explain algorithms and processes for the Rendering Pipeline. Additionally, it offers tasks the learner can solve via VR interactions. More concrete visions are listed in the following.

K12: CG basics, increasing interest

K12 classes This application could be used to explain to pupils the basic principles of Computer Graphics via VR. A side effect could be an increasing interest in Computer Graphics or Computer Science in general.

Lecture content, deepen into algorithms

Higher education Some universities offer Computer Graphics related courses and extend their learning material by VR applications. In the following are two possible cases:

- Students could use the application to get a better understanding of some Rendering Pipeline features, their relations to each other, mathematical backgrounds and necessary algorithms.

- Lecturers could get into VR application during their lecture, manipulate the scene and let the students see what they does via projector or stream to mobile VR (e.g. Google Cardboard).

Research for VR as learning technology

Research From perspective of didactics and TEL, this application may support proving or disproving of the hypothesis formalized in section 1.2: This application could be used to conduct studies to analyze learning behavior of learners in implemented scenarios to make statements about effectiveness of tasks, visualizations, interactions. Research questions could be adjusted and the importance of VR in education could be discovered more.

Extension for VR classroom

VR Classrooms Virtual environments can be extended to train pragmatical situations. One of them is the behavior training of prospective teachers. In reality the training would be very elaborate. This application may deliver useful findings for VR classrooms or create re-usable modules for the implementation of such VR classrooms.

Supports modern learning

Distance Learning In the future, students could be guided through the pipeline via mobile VR solutions. Lecturers (or teachers) could manipulate the scene and show differences via live-steam.

3.1.2 Objectives

Objectives are SMART

Goals and expected outcomes can be done in high-level formalization called *objectives*. They focus on benefits, which help to work toward the vision and solve problems instead of implementing non needed features. They differ from requirements, because they are less specific and technical. Their gain is mainly the raise of productivity and orientation [43]. Objectives are described as specific, measurable, achievable, relevant and time-bound (*SMART*).
The objective for the scope of a master thesis is:

On November 27, 2020, this project will result in an extensible Educational VR Application pilot prototype with a first version of a Guided Tour through the Computer Graphics Rendering Pipeline.

3.2 Use Cases

The above formalized vision resulted in five overall application contexts for this application: *K12 classes*, *higher education*, *research*, *VR classrooms* and *distance learning*. This section defines stakeholders, formalizes them into personas and learning scenarios and finally provides initial storyboards.

3.2.1 Stakeholders

Stakeholder researchers, students, teachers, developers.

According to Stefan et al. [64], in general the stakeholders for VREs are researchers, students and teachers (or lecturer). In fact, they match with visions in section 3.1. Additionally, it is important to consider the developer as an stakeholder, because they will extend the application in its connected systems.

3.2.2 Personas

Targeting more specific users makes the design easier [43]. One possibility to do this, is to model personas to fit the whole context range. In order to respect different knowledge levels on the stakeholder "students", it is necessary to differ between pupils and university students. From now on, "pupils" are learners in context of K12 education and "students" learners in context of higher education. Consequently the names will be kept for the following derived personas.

Significant difference between pupil and students

The personas structure is based on the template of figure 31.4 in [43]. They are organized in four quadrants: (1) the upper left has a *sketch about the person* and their name, (2) the upper right contains a *basic description* (e.g. job, experience, competencies, attitude, activities and age), (3) the lower half has *different types of challenges* (like problems, pain points, needs, desires, etc.) on the left and on (4) the right a *description about how the person relates to VR* (knowledge, dream VR system, vision of VR, hardware access, budget and VR activities).

To get realistic and evaluated personas, an online questionnaire for each target group was conducted (see chapter E). The results were evaluated quantitatively and the persona design were decided by following rules:

Personas were created by quantitative questionnaire

1. Age, semester and job experience years got the average value.
2. In personal ambitions, hobbies, competencies, traits, issues, worries the most occurred quantified value was used. If there were no unambiguously identified value, then a value was picked randomly.
3. For the reflection of interest and knowledge about Computer Science and VR, the greatest rated value was assigned to persona. In case of rate equality this criteria was skipped.
4. To reflect the VR personas relation to VR the most quantified VR device experiences, favorite VR systems and the participants' VR visions were taken.
5. All questionnaire participants have to wear glasses. To get more diversity without result distortion, sometimes "wears contact lenses" were randomly used instead of "wears glasses".
6. To keep distance from stereotypes or gender conflicts, the appearance (avatar) and gender of a persona was generated randomly by using the online avatar generator *Avataaars Generator*[1] [65]. Finally, unnecessary accessories were removed and a friendly gaze was ensured.
7. Every persona got assigned a generic names in alphabetic order.

The results can be seen in table 3.1 (pupil), 3.2 (student), 3.3 (teacher), 3.4 (researcher) and 3.5 (developer). The following tables are strictly using abbreviations CS for Computer Science, CG for Computer Graphics and VR for Virtual Reality.

[1] Avataaars assets are free for personal or commercial use (stand: 10/30/2020).

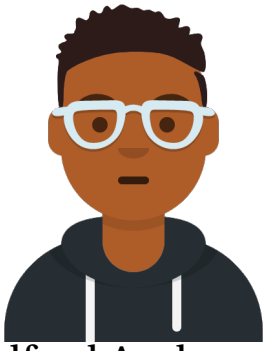
 Alfred Anderson	• **Pupil** • 15 years old • visits STEM class 9 in gymnasium • favorite color is blue • favorite animal is cat • plays soccer and computer games • meets friends often • good in painting, maths, music and programming • wants to become doctor or computer scientist
• wears glasses • has some allergies • nervous before exams	• good CS knowledge, highly interested in learning more • less relation to CG, highly interested in learning more • less knowledge about VR, tried Playstation VR once, is little bit interested in learning more • no dream VR system, no VR access, thinks VR will be more important in future

Table 3.1: Persona: Pupil

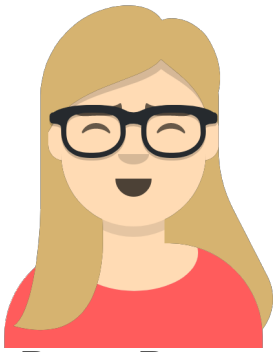
 Beate Bates	• **Student** • 22 years old • studies CS in 4th semester • subtle programmer, good listener and likes chess • likes gaming very much and does fitness • likes dogs • wants to become freelancer in software development
• wears glasses • exam fear • worries about future and failure	• very good CS knowledge, always motivated to learn more • has less knowledge in CG and VR, little bit interested in learning more • tried VR once by playing games with Oculus Rift or Mobile VR • has no favorite VR system, has no access to a system • thinks VR will be more important for simulations or games

Table 3.2: Persona: Student

 Christina Conor	• **Teacher** • 28 years old • is teacher since 3 years • wanted to be teacher since childhood • does fitness and likes dancing • speaks good english, is stress resistant, likes reading, optimistic person, very organized and patient • gets pupil excited for CS
• wears contact lenses • has hay fever, fructose intolerance and problems with knees • worries about good work-life-balance • gets dizzy easily	• very high interest for CS • highly interested in VR, but not so strongly in CG • is good in CS, had first contact with CG, tried VR once • has no favorite VR system, no access to VR and does not use it • thinks VR will be more important for simulations, games and education

Table 3.3: Persona: Teacher

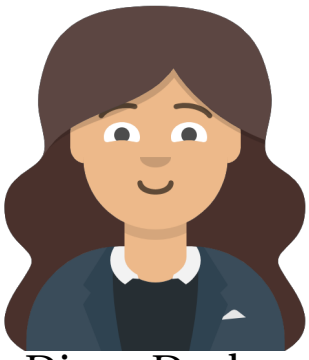
 Diana Decker	• **Researcher** • 29 years old • researcher since 3 years • research field is Learning Technologies • was working in industry, but decided to do research • likes climbing, cycling and board games • good at working with people, good sense of balance and good facts memory
• wears contact lenses • hates running • gets sun-burnt immediately • feels pressure to get doctorate • fear of isolation or loneliness • fears to not having enough time to finish all projects	• is very highly interested in CG and VR • but is medium interested in CG • professional knowledge in CS, good knowledge in CG and VR • tried a lot of different VR systems • favorite VR systems are Valve Index and HTC Vive Pro • thinks VR will be more important for education, games and simulations • has access to VR at university • is working with VR and develops apps or games for VR

Table 3.4: Persona: Researcher

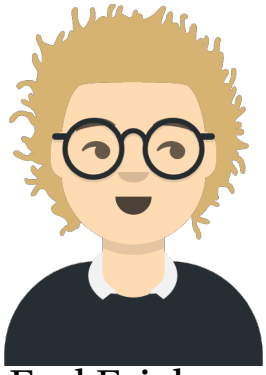
 Earl Erickson	• **Developer** • 24 years old • 6 years developer experience • develops web and mobile apps • always wanted be a developer • does programming also leisure for • is a good programmer and adaptive person • is vegan, loves challenges and cats
• wears glasses • has chronic backache • worries about Corona and its effects	• very highly interested in CS and is a professional • has medium interest in CG and VR • has good knowledge in CG and has tried VR once • has tested Oculus Rift and Mobile VR and no dream VR system • thinks VR will be more interesting for education, industry and simulations

Table 3.5: Persona: Developer

3.2.3 User Stories

INVEST: Independent, Negotiable, Valuable, Estimable, Small, Testable

User stories are short concepts or descriptions of features the stakeholders would like to see. Ideally user stories fit the *INVEST* (Independent, Negotiable, Valuable, Estimable, Small, Testable) principle [43] and are derived from interviews with the personas. But in iterative design it is recommended to discover requirements by "helping the user to find out, what they want". The persona pupil, for example, has less relation to CG and doesn't know, what to expect there. Showing them a prototype would give them a concrete context and help to discover, what they expect.

Starting with the last concluded persona (developer), a developer wants obviously to extend the system and implement new scenarios. Personal ideas (seeing me as developer) could reflect expectations to the system of a developer.

As the supervisors of this work, can represent the personas *researcher* and *teacher*, the user stories were made in their agreement. They want to primarily transfer CS knowledge to their students and analyze possible effectiveness of the transmitted learning material to improve teaching methods.

As *students* have to cover requirements provided by the *Computer Graphics* course and further related courses, the learning material shall be more detailed and complex (e.g. discovering algorithms or higher shading techniques like *ambient occlusion* or *global illumination*). Implementation of those would exceed the scope of this work. However, first user stories are made for *student* and has to be adjusted on later studies.

pupils' requirements have to be discovered

As already mentioned, a pupil may doesn't know, what they expect from the learning scenario. A valid progress is to find this out in later studies (*design-based research*) by showing them the pilot prototype and doing a post-questionnaire of what they expect afterwards. This practice is also recommended in iterative design and is called *continous discovery* [43]. Multiple iterations will approximate their real needs.

As a pupil, I ...
... want to understand how 3D geometry works.
... want to understand the purpose of the frustum.
... want to see how clipping works.
... want to understand the idea of Rendering Pipeline.
As a student, I ...
... need to understand the mathematical background of transformations in CG.
... want to understand the algorithms used in the Rendering Pipeline.
As a teacher, I ...
... want to use the app in my lecture to visualize different parts of the Rendering Pipeline.
As a researcher, I ...
... want to conduct a study to collect learner's activities and analyze the effectiveness of the RP learning scenario itself.
... want to discover the effectiveness of visualizations, interactions and tasks.
... can show this VR application to pupils to increase their interest in CS.
As a developer, I ...
... want to implement additional learning scenarios or tasks fast.

Table 3.6: User Stories

From perspective of the persona *pupil*, a RP learning scenario has to be designed on a lower complexity level to make it better understandable. The persona *pupil* visits STEM class 9 in a gymnasium and is highly interested in Computer Science. In case of someone is not so interested in CS, the VR experience of this work could ignite an interest.
To achieve a pilot prototype which can be used for requirements discoveries, first user stories will be created based on the knowledge about CG and VR from the personas. The pilot prototype shall focus at first to deliver a rough idea of the Rendering Pipeline progress. Based on future findings the complexity can be increased or decreased.
In table 3.6 the user stories were created based on previously arguments: pupils want to get an idea of the Rendering Pipeline, students want to understand more background, teachers want to integrate it in their lecture, researchers want to get findings about effectiveness of the learning scenario and finally a developer wants to extend the system.

3.2.4 Learning goals of the Guided Tour

To design a Guided Tour its' learning goals were submitted in the following by reducing the stages to the most essential parts. This section holds learning goals for each stage of the *Simplified Rendering Pipeline Model* (section 2.1.3). Each point of following list is completing the sentence "The learner shall understand in the ...". For example, the first point has to be read: "The learner shall understand in the Introduction scene that for the output they can see on their screen a RP is needed."

The learner shall understand in the ...

- **Introduction scene** that for the output they can see on their screen a RP is needed.
- **3D Geometry stage** that a 3D mesh is constructed by vertices and triangles.
- **3D Transformation stage** that the visible end result (the 2D image) depends on the settings of a virtual camera and the frustum.
- **Clipping stage** that "Clipping" is needed to not render scene objects outside the virtual camera view space.
- **Rasterization stage** that visible primitives of 3D geometry create fragments (potential pixels.
- **(Local) Lighting stage** that light defines the colors of an object and supports the impression of depth.
- **Texturing stage** that textures are 2D images, which are projected on a 3D model.
- **Visibility Test stage** that only visible fragments become pixels at all.
- **Image stage** that the 2D image is the result of all previous stages.

3.2.5 Storyboards of Learning Scenarios

Visualize 3D Geometry, 3D Transformation and Clipping at first

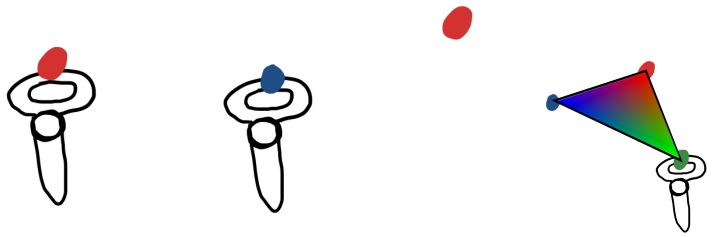
To implement concrete scenarios and to get an idea of the planned visualizations storyboards can be useful [43]. In order to not exceed the scope of this work, first "complex" visualizations were reduced to the first three stages of the *Simplified Rendering Model*.
According to VR application design principles (section 2.2) it makes sense to think about interaction metaphor and techniques here.

Volume-based selection and pointing pattern is sensible

Design decision of selection and locomotion patterns To understand the *3D Geometry* stage of the Rendering Pipeline, it makes sense to understand how to place vertices and connect them into triangles. As the user is already in 3D space, it may be intuitive to attach placed vertices onto the VR controller during the VR experience (like in section 2.3.2 *Tilt Brush*) and let the user decide where to place them. The connection of vertices to a triangle shall be done by selecting the vertices via controller. Since the geometric surface of the controller is used for interactions, this pattern is called *Volume-based selection* (table B.1). On the other side, the frustum interaction scenario needs selection from remote (e.g. planes). Thus, a *Pointing* selection makes sense. First experiences from the later conducted pilot study will show whether the choice has to be changed or not. Other possible different selection types are explained in [43] and are summarized in table B.1.
The choice of a locomotion pattern is easy. As there is already a pattern that can be used for any room size and has less VR sickness issues [43], the choice fell on the *teleportation pattern* .

experience 3D geometry

3D Geometry A 3D mesh consists of triangles connected by vertices and vertex indices. Each triangle is colored. Simple coloring can be done by assigning colors to vertices and interpolate between them on triangle's surface. To understand this progress, it could be helpful to let the learners to place vertices with different colors inside the 3D space and connect them into triangles on their own. All triangles could then result in a mesh. This way, the learner can experience how 3D meshes are constructed. In addition, the effect of color interpolation between the vertices can be seen immediately. In figure 3.1 is an storyboard example for this scenario.



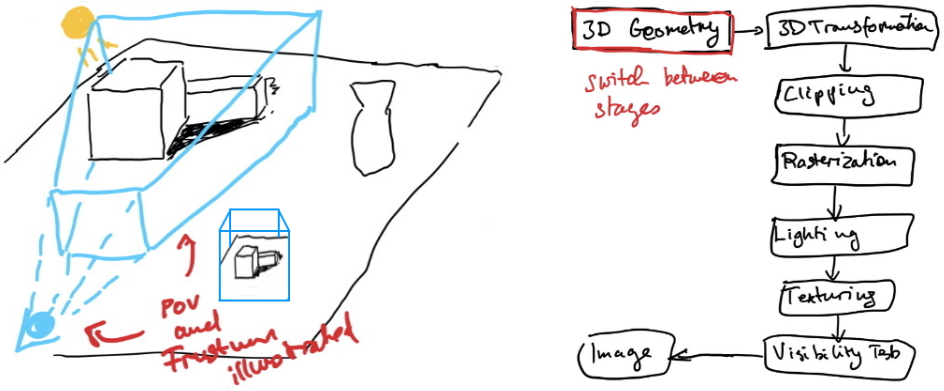
(a) Step 1: Place red vertex ***(b)*** Step 2: Place blue vertex ***(c)*** Step 3: Place green vertex and connect all to a triangle

Figure 3.1: Storyboard: 3D Geometry

Live manipulation of the frustum

3D Transformation and Clipping

Mesh transformation to normalized device coordinates and the clipping progress can be well visualized in virtual world. In figure 3.2 the learner can see the frustum and the resulting unit cube live. They shall be able to change frustum parameters live to observe the transformation directly. They can go around anytime and discover the results.
The learner is inside a virtual scene with a visualized frustum (figure 3.2(a)) and is recognizing that some objects are inside and others are outside the frustum. They can look any time at the resulting frustum unit cube placed next to the frustum and can switch to another stage any time via a menu (figure 3.2(b))



(a) Step 1: User inside the scene ***(b)*** Step 2: UI Menu

Figure 3.2: Storyboard: 3D Transformation and Clipping

3.3 Requirements

This section summarizes the previous concept fragments into concrete functional goals. To aim good quality of this work some non-functional requirements were added. The functional requirements are grouped in the sections *Must*, *Should* and *Could*. The first section contains requirements, which are mandatory to achieve. The second category contains requirements for good results for this project. The last group contains requirements which are optional, but would create a big upgrade for this application. Tasks are assigned here depending on their priority. Non-functional requirements are mostly done passively and improve the quality of the work itself.

Functional Requirements

functional requirements for content

Must

- **Educational VR Application ground**: Create a concept for a VR learning application for educational context and implement a first prototype in Unity with reusable components.
- **Perspective Transformation**: To visualize the perspective transformation the stages *3D Geometry, 3D Transformation* and *Clipping* has to be covered.
- **Collect breadcrumbs**: Collect learners breadcrumbs to collect his pathway.

Should

- **Guided Tour prototype**: Implement a Guided Tour for RP learning scenario.

Could

- **Advanced shading scenarios:** Implement advanced shading scenarios like *Backface culling, Screen Space Ambient Occlusioon* and *Screen Space Reflections*.

Non-functional Requirements

non-functional requirements for quality assurance

- **Documentation**: Code and design decisions shall be well documented in this work via comments, UML diagrams, math or pictures.
- **Expendable**: New VR learning scenarios should be added easily.
- **Maintainability**: The project must be maintained easily by the Learning Technologies Research Group. Follow best practice code guidelines (like Google JavaScript Style Guide[2] and Unity best practices[3]).
- **Modularity**: Every component or class should be as independent as possible.
- **Performance**: Real-time computation is essential in VR. As performance has influence on latency and refresh rate, the performance get measured by these metrics. The latency shall be under $100ms$ and the refresh rate between 60 Hz and 120 Hz [43].
- **Usability**: Usability should be as simple as possible. Every option should be found easily.

[2] Google JavaScript Style Guide: `https://google.github.io/styleguide/jsguide.html` (visited on 05/11/2020)
[3] Unity Best Practices Guides: `https://docs.unity3d.com/Manual/BestPracticeGuides.html` (visited on 05/11/2020)

Chapter 4 Technical Background

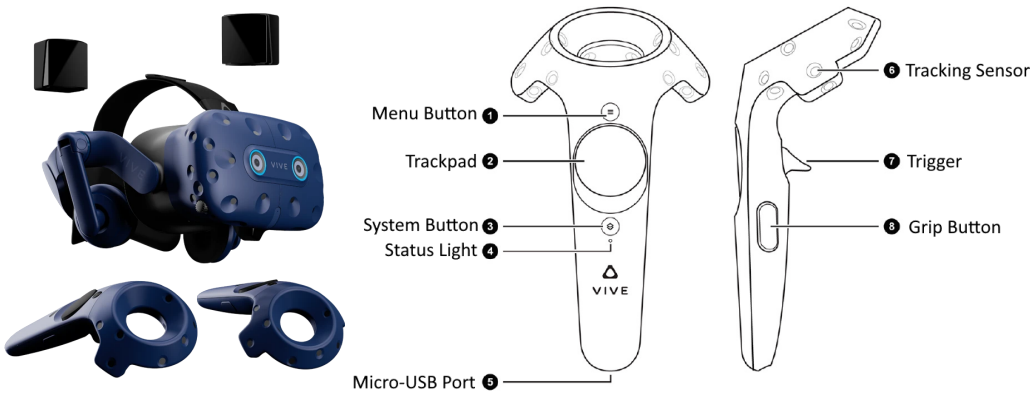
This chapter explains all technology which is needed for the implementations of this work. It starts from used hard- and software and goes over to used technologies and frameworks.

4.1 Hardware

In section 2.1.2, different equipment types to run Virtual Reality were explained. The two immersive types of VR systems are CAVE systems and HMDs. The Learning Technologies Research Group has access to a *HTC Vive Pro Eye*[1] HMD. Thus, the aimed Educational VR App was developed for this equipment.

HTC Vive Pro Eye as HMD

HTC Vive Pro Eye HTC Vive Pro Eye is a premium VR equipment for room-scale tracking and the possibility for precise eye-tracking. It satisfies image quality, ergonomics and comfort criteria very well[66]. The equipment includes two VR controllers and headphones for professional-grade sound. Two IR cameras can detect markers on the HMD and controllers and estimate a position in 3D spaces this way. The equipment devices can be seen in figure 4.1(a) (IR-cameras on the top, HMD in the middle and controllers on the bottom). Figure 4.1(b) shows a sketch of the controller with its controller trigger names.



(a) HMD and Controller. Image taken from `https://www.vive.com/`

(b) Sketch of a HTC Vive Pro Eye controller. Modified and taken from [67]

Figure 4.1: HTC Vive Pro Eye equipment

[1] HTC Vive Pro Eye: `https://www.vive.com/eu/product/vive-pro-eye` (visited on 09/20/2020)

VR needs high compatibilities power

High-Performance Computer HMDs experience quality depends on the critical quality criteria: resolution, refresh rate, field of view, lens distortions, screen door effect, ergonomics and integrated tracking system. It is obvious that the first two criteria require high computation power. As operating system, Windows 10 Education 64-bit was run on a *Schenker Compact 15*[2] laptop with an Intel Core i7-8750H CPU (2.20GHz), 64 GB RAM Memory and NVIDIA GeForce RTX 2070. This computer fulfills all system requirements (table 4.1), except for the video output port. Instead of a DisplayPort 1.2 the laptop would need a bidirectional Mini DisplayPort cable.

Component	Recommended	Minimum
Processor	Intel Core i5-4590/AMD FX 8350 equivalent or better	Intel Core i5-4590/AMD FX 8350 equivalent or better
GPU	NVIDIA GeForce GTX 1070/Quadro P5000 equivalent or better, AMD Radeon Vega 56 equivalent or better	NVIDIA GeForce GTX 970, AMD Radeon R9 290 equivalent or better
Memory	4 GB RAM or more	4 GB RAM or more
Video output	DisplayPort 1.2 or newer	DisplayPort 1.2 or newer
USB port	1x USB 3.0 or newer	1x USB 3.0 or newer
Operating system	Windows 10	Windows 7 SP1, Windows 8.1 or later, Windows 10

Table 4.1: System requirements of HTC Vive Pro[3]

4.2 Software

In context of my work the VR application requires several software. All software is free for educational purposes. *Unity* was used as the development environment of the VR application, *Microsoft Visual Studio 2019* for C# code and *JetBrains WebStorm 2020.2* and *NodeJS* for web development. The other tools were used as utilities. This chapter briefly explains the purpose of each software.

Unity is an editor for interactive 2D and 3D experiences

Unity For the implementation of the VR application, Unity[4] was used. Unity is a real-time development platform for creating interactive 2D and 3D experiences. It's commonly used for games, automotive, transportation, manufacturing, film, animation, cinematic, architecture, engineering and construction. Built-in XR platform integrations made it possible to use XR SDKs (like OpenVR[5]) to develop Virtual Reality applications with Unity. To provide better development support, Unity started to implement Unity XR[6], while deprecating Built-in XR platform integrations.

A Unity scene is composed of game objects, which contain assets and Components

Unity has, so called, *game objects*. A *game object* can be placed in the virtual scene and contains

[2] Schenker Compact 15: `https://www.schenker-tech.de/schenker-compact-15` (visited on 09/20/2020)
[4] Unity: `https://unity.com/` (visited on 09/14/2020)
[5] OpenVR SDK: `https://github.com/ValveSoftware/openvr` (visited on 09/21/2020)
[6] Unity XR: `https://docs.unity3d.com/Manual/XR.html` (visited on 09/21/2020)

children, which are *game objects*, too. A *game object* has a, so called, *transform* (representing the position, rotation and scale) and can contain *components*. A *Component* is a special form of Unity's assets (e.g. a Unity script with some custom parameters). Assets can be *3D models animations*, *prefabs* (re-useable pre-fabricated *game object*), *materials*, *scripts*, *shader* or *sounds*. The Unity editor has some non-overlapping windows (or *panels*), which can be subdivided on own choice (example in figure 4.2). A *Scene* is a preview, where the developer can visually compose the virtual scene. Here they can transform, place or delete game objects. The *Game* window shows the rendered result of the scene from the perspective of the main camera. On *Play Mode* the game is run. *Hierarchy* provides a tree view for all objects in the scene, *Project* contains all project assets and *Console* is a message board for warnings or errors. *Inspector* holds all properties of currently selected scene objects and Components can also be added here. A more detailed introduction into Unity can be found in [68].

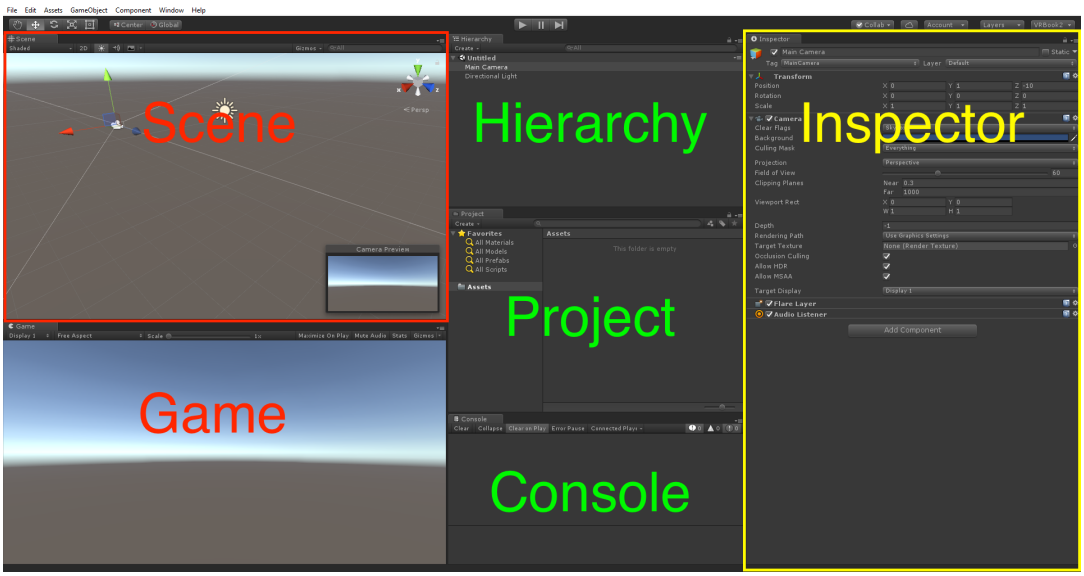


Figure 4.2: Unity Editor User Interface [68]

Microsoft Visual Studio as IDE

Microsoft Visual Studio 2019 One of many IDE's for C# is *Microsoft Visual Studio*. It has auto completion and a lot of hints for code improvements. It is suited for Unity (e.g. support of breakpoints in debug mode) and will be automatically recommended on Unity's setup.

WebStorm for web technologies

JetBrains WebStorm 2020.2 Some parts of this thesis needed to be programmed with Web Technologies (HTML5, CSS3 and JavaScript). For this purpose JetBrains WebStorm 2020.2 was used. It has many file templates, tool plugins, easy version control and has a free students license. Furthermore, personal experience with it made comparisons not needed.

NodeJS as micro-service platform

NodeJS and Node Package Manager NodeJS[7] is a JavaScript runtime environment based on Chrome's V8 JavaScript engine. With NodeJS it is possible to implement scalable micro-services or server applications. Big global companies use it in production with big benefits[69]. Implementations are packed into so called *node modules*. This way it is easy to reuse them. A developer can include packages for their implementation inside the *package.json* file of their project.

[7] NodeJS: `https://nodejs.org/` (visited on 09/21/2020)

NodeJS delivers a so called *Node Package Manager* (NPM)[8] for downloading and installing these NodeJS modules.

Postman for REST requests

Postman *Postman*[9] is an useful tool for working with HTTP requests. It is very helpful for *RESTful API*s (short REST API). REST (*REpresentational State Transfer*) is a stateless architecture to manage distributed systems [70]. The nature of REST APIs allows to create, retrieve, update or delete resources stateless over HTTP protocol.

Docker for shipment

Docker *Docker*[10] allows to package software (with all its dependencies and settings) into standardized units (*container*) for easy development, shipment and deployment. This allows to run the software on any computing environment, where Docker is used. Every container is strongly isolated and very lightweight.

version control of files

GitLab, GitHub and Git LFS *GitLab*[11] is a complete DevOps[12] platform, where developers can use git version control, organize issues, discuss solutions in "merge requests". GitLab repositories can be made public or private. GitHub[13] is similar to GitLab with some differences, which are not important here. GitHub was mainly used to fork or clone from open third party projects. By checking out files from git, there may be problems with transferring larger files. Git LFS[14] can help out here. It created a pointer to larger files, which will be checked out together with all files. Afterwards the large files can be downloaded via Git LFS separately.

VR support via Steam

Steam *Steam*[15] is a video game digital distribution server by Valve. Users can buy online keys of games, add contacts, communicate and play together with them. But Steam has some more functionalities, like streaming, input device interface and VR support.

4.3 Used Technologies and Frameworks

The implementation involves the usage of several technologies. This chapter contains details about each used technology to provide transparency about the decision to use them.

xAPI statements defined by actor, verb and object

Experience API (xAPI) The *Experience API* (xAPI)[16] was developed by an open community led by ADL (Advanced Distributed Learning Initiative)[17]. The xAPI specification allows to describe user activities in form of *statements*[18]. A *statement* has an *actor*, *verb(action)* and *object* structured (for example) in JavaScript Object Notation (JSON) format[19]. Verbs or objects are strictly defined and are uniquely identified (*id*) by *IRI (Internationalized Resource Identifier)*[20]. Sometimes the difference between a verb or object is not clear. This is why it can be sensible to define

[8] Node Package Manager: `https://www.npmjs.com/` (visited on 09/28/2020)
[9] Postman: `https://www.postman.com/` (visited on 09/28/2020)
[10] Docker: `https://www.docker.com/` (visited on 09/28/2020)
[11] GitLab: `https://gitlab.com/` (visited on 09/28/2020)
[12] DevOps: `https://aws.amazon.com/devops/what-is-devops` (visited on 09/28/2020)
[13] GitHub: `https://github.com/` (visited on 09/30/2020)
[14] Git LFS: `https://git-lfs.github.com/` (visited on 10/06/2020)
[15] Steam: `https://steampowered.com/` (visited on 09/30/2020)
[16] Experience API (xAPI): `https://xapi.com/` (visited on 08/17/2020)
[17] Advanced Distributed Learning Initiative: `https://adlnet.gov/` (visited on 08/17/2020)
[18] xAPI Statements: `https://xapi.com/statements-101/` (visited on 08/17/2020)
[19] JavaScript Object Notation (JSON): `https://www.json.org/` (visited on 08/17/2020)
[20] Internationalized Resource Identifiers: `https://tools.ietf.org/html/rfc3987` (visited on 11/03/2020)

activities. Every statement can have some measured outcomes (activity has been completed, actor's score, etc.). This information is contained in the *result* field. The definition of a *context* is also possible. The context field holds information about the instructor or more details about the applied context. So called *extensions* can be used to add more information about an activity, a context or result. The statements are collected in a sequential ordered database called *Learning Record Store (LRS)*. It is also possible to define new vocabularies without any constraints and expand xAPI to new specifications [71]. The definition of new vocabularies is called *xAPI Profile*. A small example of a xAPI statement can be seen in listing 4.1. An example with more xAPI specifications can be seen in listing C.1 and more details in the master thesis by A. Brocker [67].

```
{
  "actor": {
    "name": "Sally Glider",
    "mbox": "mailto:sally@example.com"
  },
  "verb": {
    "id": "http://adlnet.gov/expapi/verbs/experienced",
    "display": { "en-US": "experienced" }
  },
  "object": {
    "id": "http://example.com/activities/solo-hang-gliding",
    "definition": {
      "name": { "en-US": "Solo Hang Gliding" }
    }
  }
}
```

Listing 4.1: xAPI statement example from `https://xapi.com/statements-101/`

Database for xAPI statements

Learning Record Store (LRS) According to the xAPI website a *Learning Record Store (LRS)*[21] is "the heart of any xAPI ecosystem". It receives, stores and returns xAPI statements. A LRS can be used as central store for different applications. According to the xAPI specification [72] it is a storage which is responsible for providing, storing or receiving access to learning records.

Utilities and panel for LRS

Learning Locker Learning Locker[22] is an open source Learning Data Warehouse application which enables architects to deliver enterprise-level data management for big learning data. Any learning or activity experience can be recorded using xAPI standard (via webhooks). Data can be aggregated, queried on-demand and rules can be defined to trigger data rules. An aggregation pipeline can be customized to export files out-of-the-box.

Vue is a modern web technology

Vue *Vue*[23] (or *Vue.js*) is a progressive JavaScript framework for developing frontend web applications. Vue can be compared to Angular and React. Vue is a lightweight and versatile framework with a high learning curve and increasing popularity [73]. The whole app consists of components which the developer can compose together to a fast loading virtual DOM application. A component is divided into a <*template*> (for HTML), <*script*> (for JS) and <*style*> (for CSS)

[21] Learning Record Store (LRS): `https://xapi.com/learning-record-store/` (visited on 09/14/2020)
[22] Learning Locker: `https://docs.learninglocker.net/` (visited on 08/17/2020)
[23] Vue.js: `https://vuejs.org/` (visited on 09/28/2020)

block. Every script block returns a JSON object with a predefined properties. Each Vue component has its own lifecycle of these properties (which can be seen in Vue documentation). This makes it easy to understand, in which order Vue properties are used (e.g. "created" is called once before "mounted"). The template block of a Vue component can be used to bind entries from the *<script>* properties or define render logic. And obviously *<style>* can be used to define *CSS* (or CSS pre-processor[24]) code. More details can be read in Vue documentation.

high learning curve, increasing popularity

Vuetify *Vuetify*[25] is a Vue based UI framework and is following the Material Design specification[26] with a lot of UI components, good support and documentation. It can be easily used in Vue applications without big effort.

Modern designed elements for Vue

Nuxt.js *Nuxt.js*[27] delivers an easily configurable, performant and modular application ground. The developer can develop a Vue application without thinking about complex configurations. For example they can decide inside of *nuxt.config.js* whether it is a *Single Page Application* or *Server Side Rendered*, the application shall use *HTTPS* protocol or what *Nuxt.js* modules are used (e.g. Vuetify). Nuxt.js is automatically using pre-configured Webpack[28] with *ESLint*[29] for *ECMAScript* and is easy to deploy.

Pre-configured architecture and toolkit for Vue

Newtonsoft Json.NET *Newtonsoft Json.NET*[30] is a high-performance JSON framework for .NET framework. C# classes can be easily serialized or deserialized to or from JSON string. It is also possible to convert between XML and JSON.

serialize JSON from and to C#

SteamVR *SteamVR*[31] is a tool for easily communicating with Valve Index, HTC Vive, Oculus Rift and Windows-Mixed-Reality-Headsets. It is available via Steam and was developed by Valve. An Unity plugin helps to address SteamVR functionality with C# in Unity. The plugin can be downloaded in Unity's Asset Store.

VR hardware interface

Learning Analytics Data Collector for VR To build a technological foundation for Learning Analytics in Virtual Reality, Annabell Brocker has created, in scope of her master thesis *A Learning Analytics Data Collecor for Virtual Reality Using Unity and xAPI* [67], an Unity plugin, a web cockpit in *React*[32] and NodeJS web socket server. The plugin contains a Unity prefab where developers can configure LRS and start sending xAPI statements to LRS. This was solved by using by using the *TinCan.Net*[33] API. In her work she implemented the recognition of head nodding (or shaking) and eye blinks (or focus) and tracked these actions in xAPI. A. Brocker has used web sockets to establish the communication between VR client and a web UI, where Unity game objects and VR controller actions can be configured, which has to be recorded in Learning Locker. More details about background, frameworks and implementation can be read in her thesis [67].

use a existing collector for VR data

[24] CSS Pre-processors in Vue: `https://vue-loader.vuejs.org/guide/pre-processors.html` (visited on 11/10/2020)
[25] Vuetify: `https://vuetifyjs.com/` (visited on 09/28/2020)
[26] Material Design: `https://material.io/` (visited on 09/30/2020)
[27] Nuxt.js: `https://nuxtjs.org/` (visited on 09/28/2020)
[28] Webpack: `https://webpack.js.org/` (visited on 10/06/2020)
[29] ESLint: `https://eslint.org/` (visited on 11/26/2020)
[30] Newtonsoft Json.NET: `https://www.newtonsoft.com/` (visited on 09/30/2020)
[31] SteamVR: `https://store.steampowered.com/app/250820/SteamVR` (visited on 09/30/2020)
[32] React: `https://reactjs.org/` (visited on 11/26/2020)
[33] TinCan.Net: `https://github.com/RusticiSoftware/TinCan.NET` (visited on 09/30/2020)

Upgrade web sockets to Socket.IO

Web Sockets with Socket.IO As the web sockets implementation in A. Brocker's work [67] was used in it's plain form, it may be more sensible to use a framework instead. One possibility for simple web socket usage is *Socket.IO*[34]. It is not only easy to use, it has also protocol fallbacks. If (for example) a browser is not supporting web sockets, Socket.IO then uses HTTP polls instead (sending HTTP in an interval). For Unity a free plugin from the Unity Asset Store (with MIT license) was used. More technical details about this whole part can be found in section 5.4.

Fast HTTP server platform

Fastify *Fastify*[35]is a highly performant alternative to the well known Node package Hapi (based on *express*). It is TypeScript ready, developer friendly and easy extendible. Fastify has an own "Ecosystem", where developers can find compatible plugins for fast creation of HTTP servers.

Unit tests with NUnit and mocha

NUnit and mocha One method to test software is *Unit testing* [74]. With *Unit tests* single methods can be tested on their correctness. *NUnit*[36]was used to execute *Unit tests* for C# code and *mocha*[37] for *NodeJS*.

Unity Timeline organizes assets at specific time

Unity Timeline *Unity Timeline*[38]is a Unity package to create cinematic content, gameplay sequences and complex particle effects. After installing this package it can be accessed via menu item *Window* > *Sequencing* > *Timeline* in Unity. The package window (figure 4.3) contains a list of timeline assets and a time bar for each element to regulate sequences. Every timeline asset has its own functionality and can be be composed into a cinematic sequence. For example: one asset is for the regulation of character animation and one other asset is to play a sound. Each asset has a duration and start time. Unity timeline has the potential to be used for creating a Guided Tour. Sequences can be regulated and missing assets extended.

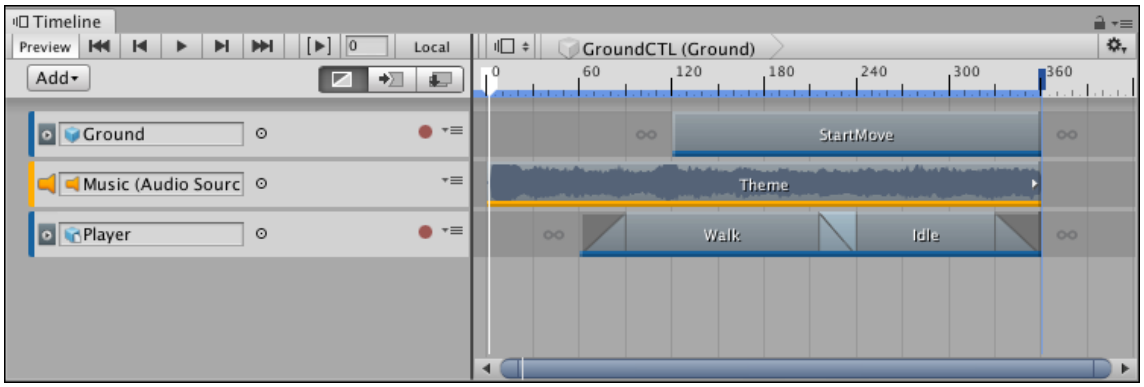


Figure 4.3: Unity Timeline window. Picture taken from `unity3d.com`.

[34] Socket.IO: `https://socket.io/` (visited on 09/30/2020)
[35] Fastify: `https://www.fastify.io/` (visited on 10/04/2020)
[36] NUnit: `https://www.nuget.org/packages/NUnit/` (visited on 11/25/2020)
[37] mocha: `https://www.npmjs.com/package/mocha` (visited on 11/25/2020)
[38] Unity Timeline: `https://docs.unity3d.com/Packages/com.unity.timeline@1.2/manual/index.html` (visited on 10/25/2020)

Chapter 5 Realization

system overview in following

As part of Learning Analytics in any application, it is necessary to think about *how* the effectiveness of the learning materials can be analyzed [22]. According to [23] one option is to observe user's activity breadcrumbs. With combination of user feedback, those can be used to analyze activities on their effectiveness or uselessness. The Learning Technologies Research Group had already started to implement a technology [67] for this case. In scope of this thesis, the needs for the data collector increased. Thus, part of this work was to extend their technology with functionalities and additional tools. Figure 5.1 shows the overall resulted system and technology stack of this work. After a system overview, this chapter will go through each sub-system, discuss design decisions, explain implementations and provide result previews.

needs to the VR Data Collector increased

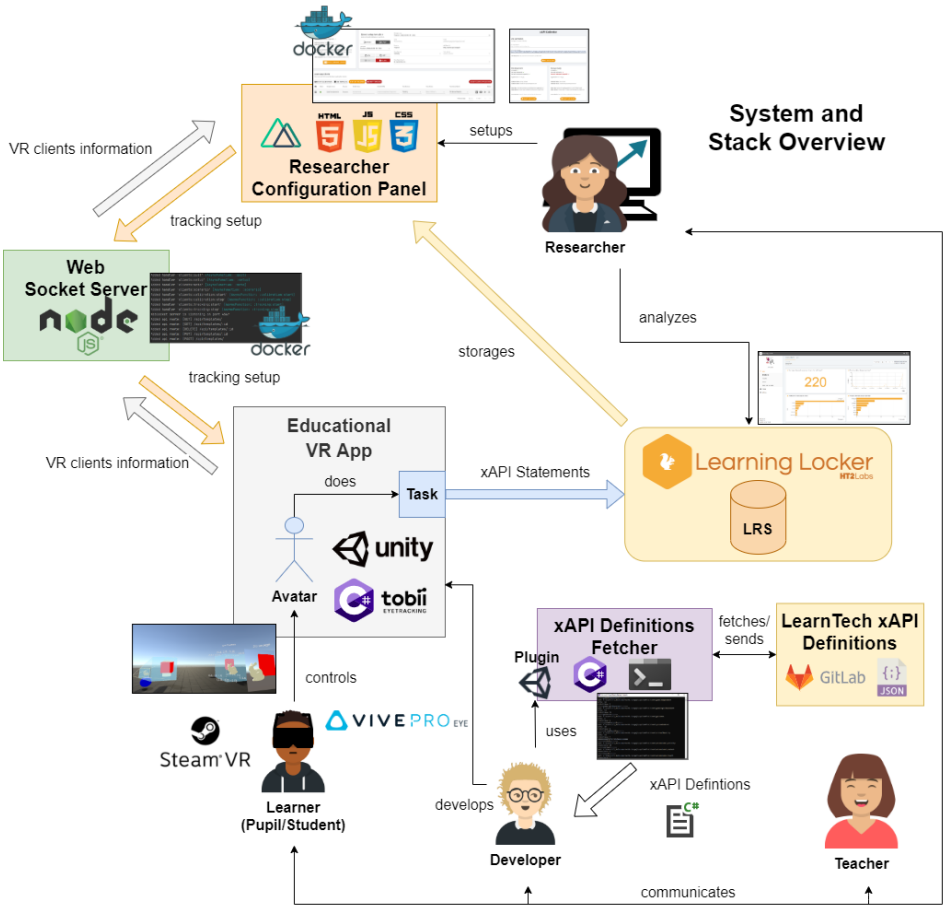


Figure 5.1: System and stack overview, made with draw.io[1].

[1] All trademarks displayed on this slide are the trademarks of their respective owners.

5.1 System Overview

learner activity are tracked on Learning Locker

The whole system can be divided into a *Learning Record Store* and following software components: the Educational VR Application (section 5.3), *LA Data Collector Panel and Server* (section 5.4), *xAPI Definitions* (section 5.5.1) and *xAPI Definitions Fetcher* (section 5.5).
A learner (pupil or student) uses a VR equipment to dive into a Virtual Reality scene. They controls an avatar, who is interacting with virtual objects. Through virtual interactions the learner can solve tasks. After appearance inside the virtual scene, the VR application connects to *Web Socket Server* and sends information about its' scenario. The socket server distributes all its knowledge to all connected *Researcher Configuration Panel* instances. Those are web applications, which can be used by researchers to regulate the tracking behavior and sessions of a VR scene. Any VR client reacts immediately on its' configurations. During a "tracking session" a VR client is sending xAPI statements to its' selected LRS. Afterwards researchers can observe stored statements on *Learning Locker* to analyze them.

what to track is configured in researcher's panel

Developers use Unity to develop virtual scenes. Certainly, they have to implement the xAPI statements for the learning tasks. Standardized xAPI verbs, activities and extensions can be taken from the *LearnTech xAPI* repository. To avoid working with a hard-coded xAPI vocabulary, C# classes can be generated to define the statements over static C# fields and methods. The *xAPI Definitions Fetcher* translates these classes from JSON objects, which are placed on a GitLab repository. This tool is a C# console application and can be called via a Unity plugin directly from Unity's editor.

5.2 Workflow and Communication

live communication is ensured

The original idea of the live communication between the Unity client and the browser client is still the same. But the communication was improved with some more foreground and background events. Figure 5.2 represents the whole sequenced workflow of the whole system. At first, the panel request LRS clients from *LRS REST API* via *Authorization* in *HTTP header*. The *LRS REST API* deliver valid LRS client the researcher can use. After selecting a *LRS client* on the *Researcher Configuration Panel*, the *LRS REST API* delivers "access" to this client. At this point the panel connect to *web socket server* and establish communication with all Unity clients. The *web socket server* sends frequently a list of the active Unity clients to the researcher's panel. The researcher can start (or stop) a calibration or tracking session of individual Unity clients through their panel anytime. The aimed Unity clients react on these operations and start the called session. While a tracking session is running, the concerned Unity client is transmitting xAPI statements to the selected LRS client.

selection of LRS clients

multiple clients are possible

control Unity over panel

5.3 Educational VR Application: Rendering Pipeline

re-usable assets were stored in the namespace EducationalVR

The *Educational VR App: Rendering Pipeline* is a prototype of an Educational VR Application for experiencing the Rendering Pipeline in VR. The implementation of this software includes a first Guided Tour pilot prototype to ensure primary the VW class *exposition* and partly the class *experimental*. During the implementation Unity assets were created, which could be useful for new Educational VR Applications. All reusable assets were collected to a *EducationalVR* namespace (more in section 5.3). The relation between the systems and their third-party components, can be seen in the component diagram in figure D.1. More details about all relevant included components are be described in next sections.

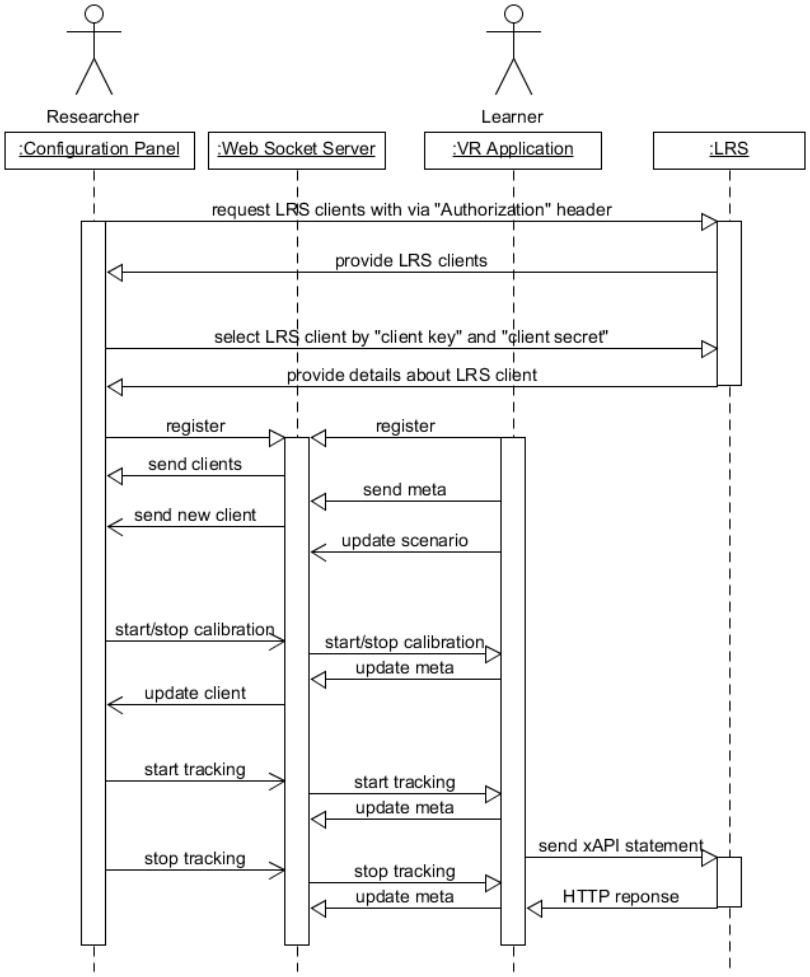


Figure 5.2: Communication sequence between clients

5.3.1 Namespace EducationalVR

has potential useful Unity scripts and prefabs

The namespace *EducationalVR* is a passive result of the development of the Educational VR Application of this work. It is not strictly planned (like an engine), and is far away from being a generic library for Educational VR Applications. But it conjuncts useful Unity scripts and prefabs for educational purposes. This section explains design decisions and the most important Unity *prefabs*.

pointing pattern

Selection of interaction pattern The different interaction techniques and metaphors were described in section 2.1.2. As the Rendering Pipeline is not a realistic scenario, the pointing pattern seems to be the most sensible here. To ensure precise interactions, it is important to not place interact-able objects far away from the actor.

prefab Player is useful

Decision for SteamVR 2.0 As SteamVR has a high compatibility to HTC Vive, the work uses SteamVR's library. The Unity prefab *Player* is already supporting a lot of functionality concerning camera, interaction and teleport.

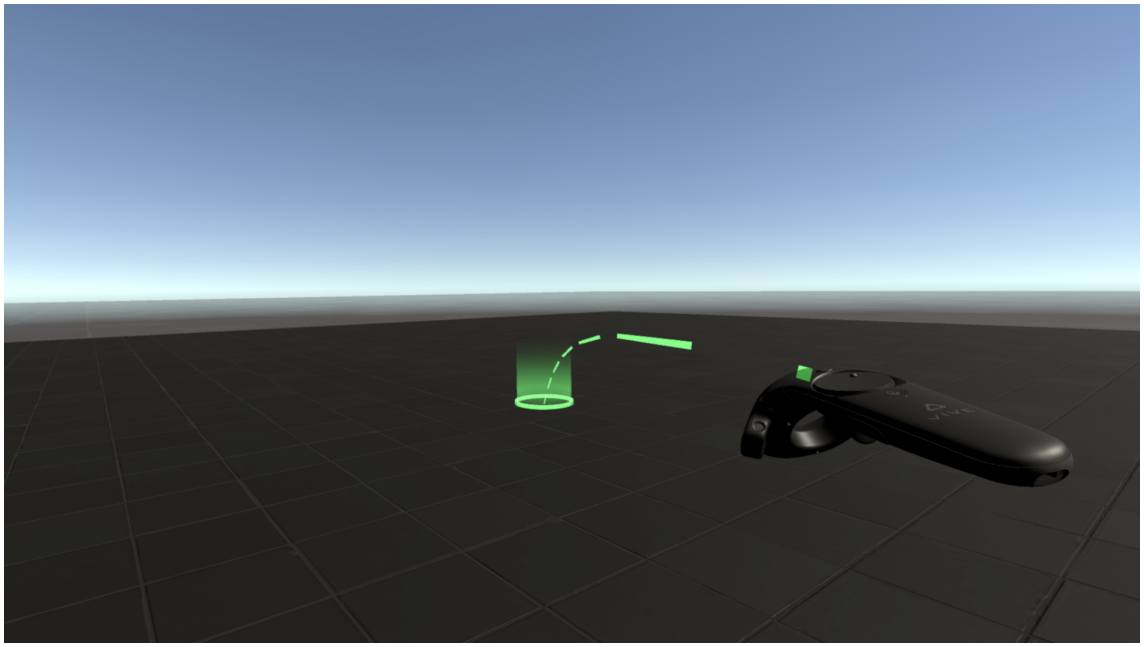


Figure 5.3: Default learning environment

VRTK has been postponed

Decision against VRTK One toolkit for VR development is *Virtual Reality Toolkit (VRTK)*[2]. It has a bride palette of Unity prefabs for interaction and UI. As it has no official support for SteamVR 2.0, the migration needs more time investment and requires code changes in the implemented interaction. Thus, the integration of VRTK was postponed.

a large neutral environment

Learning Environment To have a "neutral" learning environment, a learning environment with enough space to move around and with a visible horizon was created. Constricted virtual rooms were avoided. The environment ground conists of a not passable plane with gray floor texture (from SteamVR's assets) and a passable (via SteamVR's teleport function) one with dark floor texture (figure 5.3)[3] above the other one.

singleton game object for the learner

Learner The prefab *Learner* was created to have a main Unity game object for learners' functionalities and settings. This Unity game object is holding a singleton Unity script *Learner* with options for learner the developer can set (for example: setup of left or right hand as primary hand).

"ready-to-use" prefab for LA in VR

"Ready-to-use" prefab The previously mentioned prefab Player has been wrapped around by the prefab *Learner* and was moved into prefab *Learning Environment*. Thus, the developer can start with an Educational VR Application, by just dragging the prefab *Learning Environment* into the Unity scene.

interactable panel was needed

Dialog panel The Guided Tour prototype needed a panel to display texts and enable VR menus. The appearance of the implemented panel is based on SteamVR's Unity interaction example scene[4]. This contains a changeable text, which was used for dialogue texts. Further,

[2] VRTK: `https://www.vrtk.io/` (visited on 10/20/2020)
[3] The texture was taken from SteamVR assets.
[4] The Unity scene can be found in SteamVR/InteractionSystem/Samples/Interaction_Example.unity

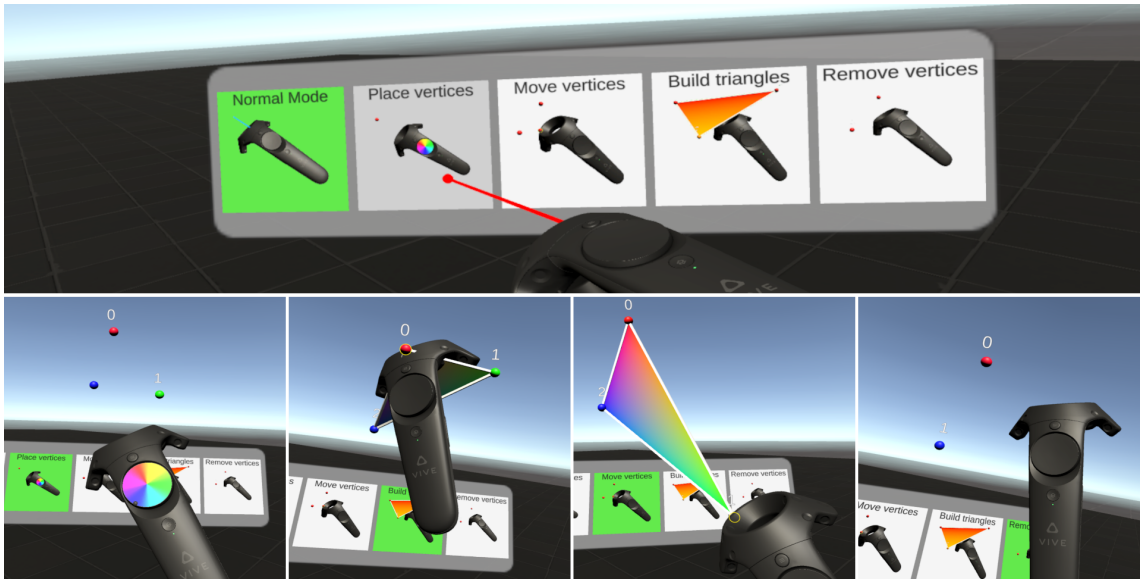


Figure 5.4: Triangle modes menu on the top. Triangle interaction modes: place vertices, build triangles, move vertices and remove vertices (from left to the right)

the panel was extended by an attached Unity UI button, the pointer can interact with. To handle the panel's position and orientation, two Unity scripts were developed: *Rotate To Transform* and *Dialog Attacher*. Both need an Unity *Transform* instance to work with. The first one simply rotates the parent game object in direction to applied *Transform* instance. The second awaits a position $y \in \mathbb{R}$, distance $d \in \mathbb{R}$ and degree $\psi \in [-360°, 360°]$. With trigonometry, it is then easy to compute panel's position $p = (cos\phi \cdot d, y, sin\phi \cdot d) \in \mathbb{R}^3$, where $\phi = \psi \cdot \frac{\pi}{180°}$. As a UI button of Unity is only working for mouse actions, the interaction with pointer had to be ensured. To achieve this, the Unity script *Pointable UI* was created. This creates a box collision area in button's shape and simply takes the already existing 3D collision mechanism between laser pointer and environment. In combination with script *Menu Panel*, many buttons can be taken into a group and can be handled by events and references (e.g. the triangle modes menu in figure 5.4).

PointableUI was made to enable pointer interaction with Unity buttons

5.3.2 Stage 3D Geometry

vertices modes place, move, remove, build triangle

One learning scenario for pupils in section 3.2.5 is the experience of geometry basics. Here, the learner shall be able to place vertices in VR and connect them into triangles (figure 3.1). But editing or removing them would be important, too. Thus, it is necessary to distinguish between five modes: *normal*[5], *place vertices*, *move vertices*, *build triangles* and *remove vertices* and to implement a main component for the mesh building. The detailed realization is described in the following.

main component for this feature

Triangle Builder The *Triangle Builder* contains a list about all created vertex objects and reacts on the active interaction mode (place vertices, build triangle, move vertices and remove vertices). To switch between these modes a panel was created, which was attached to follow the learner (figure 5.4). The learner can switch between modes by interacting with the panel by using controller's pointer.

[5] No geometry operation is active.

vertex color can be changed

Place vertices The learner can select a color on a gradient attached to their controller's trackpad (figure 4.1(b)). To get the selected color from user's thumb, the color gradient is attached to SteamVR's controller model game object "trackpad" (figure 4.1(b)). It contains a Unity *Transform* about actual finger position $t = (t_x, t_y), t \in \mathbb{R}^2$ on the trackpad. The center of the trackpad is $(0,0)$ and the range is going from $(-1,-1)$ (right, bottom) to $(1,1)$ (right, top). This makes it easy to pick a pixel color of arbitrary texture (representing a color gradient) with width w and height h by using function $color(t_x, t_y) = (t_x \cdot \frac{w}{2} + \frac{w}{2}, t_y \cdot \frac{h}{2} + \frac{h}{2})$. The touch position on the trackpad will be simply transformed into a pixel position on gradient's texture outgoing from the center. After successfully choosing a vertex color, the user can place the colored visualized vertex, by using the controller's *trigger* (figure 4.1(b)) under the index finger. A vertex representation uses controller's 3D position added by an offset to be placed in 3D space. Finally, the learner can interact with these vertex game objects to connect them to triangles.

vertex position is defined by controller's position

Build triangle To enable collision between controller's mesh and the vertex representations, a Unity *Mesh Collider* component was enabled for controller and *Sphere Collider* for each vertex game object. To avoid occlusion of these vertices by controller, an outline was added for each vertex game object with the Unity package *Quick Outline*[6]. On collision, the outline is colored yellow. By pressing the controller's *trigger* action, the collided vertex game object will be added to a list and gets a blue outline. The two last selected vertex game objects are always connected by a wire representing an edge between two vertices. Every wire holds itself the information which vertices (*from*, *to* properties in *VertexPair*) the wire connects. After two vertex game objects has been connected to each other, both get the Unity Component *VertexHelper*. This Component holds lists about incoming and outgoing wires and it's overall connected triangles. These components (*VertexPair* and *VertexHelper*) enable fast access time for further manipulation. Finally, when three vertex game objects are connected together, a triangle game object is generated, containing a mesh of three vertices with according color and position from its connected vertex game objects. A triangle game object holds the component *TriangleHelper* with all connected wires and vertices game objects.

outline on selection

two vertices ⇒ wire

three vertices ⇒ triangle

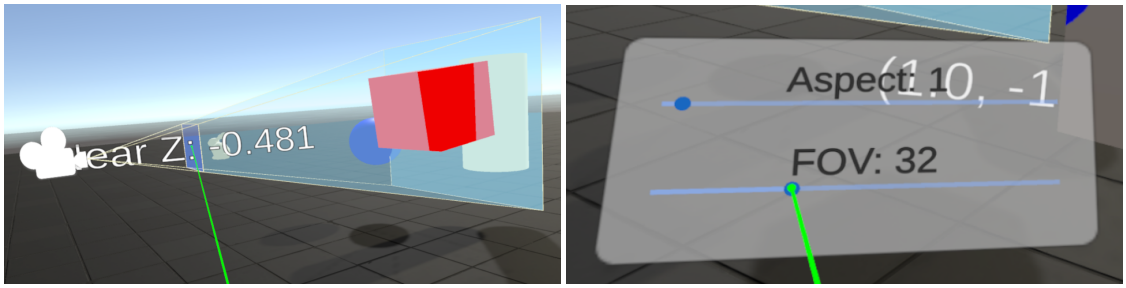
Move vertices Similar to "build triangle" mode, in this mode the collision with vertex game objects is enabled. By pressing the controller's *trigger* button (figure 4.1(b)), the collided vertex sticks to controller's mesh its' collision touch point and moves together with it. At the same time, connected wires and triangle meshes will be updated efficiently through previously mentioned helper components on the affected vertex.

sticks on touch point

Remove vertices The first idea here was to remove triangles. But, as Unity has no "ready to use" colliders for triangles and even the *Mesh Collider* didn't work here, the easiest way to destroy triangle was to remove its vertices. A vertex game object will be destroyed on pressing the controller's *trigger* button (figure 4.1(b))). As a consequence, all connected triangles and wires will be destroyed, too.

destroy triangle by removing vertices

[6] Quick Outline: `https://assetstore.unity.com/packages/tools/particles-effects/quick-outline-115488` (visited on 10/19/2020)



(a) Drag near plane by pointing on it and pressing trigger

(b) Adjust aspect ratio and field of view by using sliders on a UI panel

Figure 5.5: Changing frustum values with interactions

Implementation result Figure 5.4 shows the menu itself on the top of the picture and in the bottom all triangle modes can be previewed. To switch between these modes, the learner can select a mode via pointer on the menu. To have this menu always next to the learner, this menu was attached relatively to the player in 45 degrees to the left. As a vertex consists of an abstract 3D point and a color, it cannot be seen without any geometry. Thus, as representation a colored 3D sphere was used instead. Even though, a sphere is not an *exact* representation of a vertex, it may be enough to transfer the basic idea of this concept. Normally a wire is also invisible. But for visualization: a wire was simply build by a thin stretched white cube. A created triangle mesh uses a simple shader for the interpolation between colors. Finally, a label was placed above every vertex to illustrate the index of a vertex. In summary, the learner can experience the generation of 3D meshes by placing/moving/deleting vertices and connecting them to triangles, which result in a mesh. From the perspective of the *Simplified Rendering Pipeline*, it represents part of the conceptual *3D Geometry Stage*.

3D Geometry can be experiences by building meshes

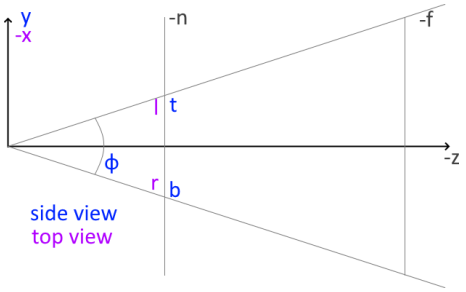
5.3.3 Stages 3D Transformation and Clipping

One of the main features of the Rendering Pipeline is the transformation from a 3D scene into a 2D image. This consists of several steps. By choosing a perspective projection, the transformation can be done with the help of a *frustum* (section 2.1.3). To make the frustum transformation understandable for the learner, it may makes sense to let the learner manipulate the frustum's values and see the results in real-time (figure 3.2(a)). The shape of a frustum depends on the values of the near plane, far plane, aspect ratio and field of view. Thus, the component *Frustum* was generated, where those values can be passed. This component generates the frustums mesh (for visualization) based on these values.

main component: frustum

Far/Near plane interaction As the far and near plane have to be interactable (needs collision detection), Unity's *Planes* were used for them, including an own game object (figure D.6(b)) and *Mesh Collider*. During the interaction by a pointer, the far and near plane is riding along the z axis (figure 5.5(a)). To implement this interaction the Unity Component *FrustumPlaneInteraction* (class diagramm in figure D.2) was created. It's simply takes the z component of the intersection point between plane and pointer (projection ray on plane) and updates the local z coordinate of its' game object. This way, the selected plane can follow the pointer.

far and near plane use Unity's plane for interaction



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

Figure 5.6: Perspective projection matrix P[1], with field of view radians $\Phi \in [0, 2\pi]$, aspect ratio $a \in \mathbb{R}$ (e.g. image size relation $\frac{\text{width}}{\text{height}}$), $t = z \cdot tan\frac{\Phi}{2}$, $b = -t$, $r = a \cdot t$, $l = -r$

adjust FOV and aspect ratio via pointer and a slider

Field of view and aspect ratio adjustment A possible solution for adjusting field of view (FOV)[7] and aspect ratio[8] may be to set the values according to the controller's orientation (rotations) in 3D space. But for first prototype already chosen selection pattern (pointing) was consequently used. Thus, a so called *Slider Panel* was implemented for the regulation both values. The slider reacts to pointer's interaction and follows the pointer's slider panel intersection point on the x axis. This value is clamped between the floating values from 0 (leftest position) to 1 (rightest position). Depending on the frustum's maximum aspect ratio and FOV values (x_{max}), it updates the current frustum (aspect or FOV) value to $f(x, x_{max}) = x \cdot x_{max}$. Both sliders can be seen in 5.5(b).

transformation on CPU side to enable interaction

Perspective transformation The two major Unity scripts for perspective transformation are *Frustum* and *Unit Frustum* (figure D.6(a)). The first one consumes provided values and computes all needed values to generate the frustum mesh. The second component uses methods from *Frustum* (class diagram in D.2) to generate a perspective projection matrix P[9] from figure 5.6. The matrix is calculated by Unity's static function *Matrix4x4.Frustum*[10]. Normally, the transformation is done on GPU's vertex shader. But to enable learners' interaction with resulted transformed mesh (and track the interactions), the transformation was done via Unity script code (running on CPU). The implemented function with commented steps can be seen in listing 5.1. As the game objects' positions are already relative to the representing frustum camera, the TRS matrix of each frustum scene object is already in eye space. After applying the perspective transformation, each deformed mesh vertex has to be transformed again to its' local space (line 10 in listing 5.1). This has to be done for a correct display of the further rendering progress of Unity. To just transform a selected group of game objects, the transformation is only applied on a copy of all children of a game object called *Scene* (figure D.6(b)). A container node called *Scene(Clone)* will be generated automatically with holding a generated unit cube around the transformed meshes.

Optimized transformation performance issue The application of the frustum transformation has a performance lack (refresh rate between 28 and 34 FPS instead of around 120 FPS). This was noticed via Unity's stats window. It is maybe caused because of all matrix multiplications for the mesh are done on CPU and not on GPU. A solution could be to do the transformation

[7] Field of view is a restriction of what is visible.
[8] The aspect ratio is mostly defined by canvas width and height.
[9] This is the projection matrix of Unity and OpenGL. DirectX' P has a negated third column.
[10] Matrix4x4.Frustum: `https://docs.unity3d.com/ScriptReference/Matrix4x4.Frustum.html` (visited on 10/20/2020)

via shaders. But as it was planned to collect learner's breadcrumbs, it made sense to make the transformed objects pointable by laser pointer and track this interaction. Thus, the performance was improved a bit on CPU side: The transformations are only updated in a fixed *Coroutine*[11] of 5 milliseconds, when the user is changing or updating a frustum value. To also save performance on updating the mesh collider of an object, this update is done with an interval of 1 second. This interval is enough, because a user will not point faster to a transformed object after changing the frustum. As result the refresh rate has been increased to 58 to 64 FPS.

Coroutines make the performance lack less noticable

Clipping The *Clipping Stage* discards 3D primitives, which will not be rendered. A not so trivial method to achieve this, is to "cut" the geometry on CPU side along the unit cube planes, create new vertices on the boundaries and finally connect vertices to sensible triangles of the mesh.But the easier method is to realize it on GPU side with fragment shader (or Unity's surface shader[12]). To do this an Unity component with shader (*FrustumObject*) was created. It simply checks the primitives against a unit cube. Depending on a component's configuration, matched fragments become invisible (figure 5.7(a)) or tinted (figure 5.7(b)). Both modes can be toggled by pressing the menu button on the user's controller.

Clipping is realized via Shader

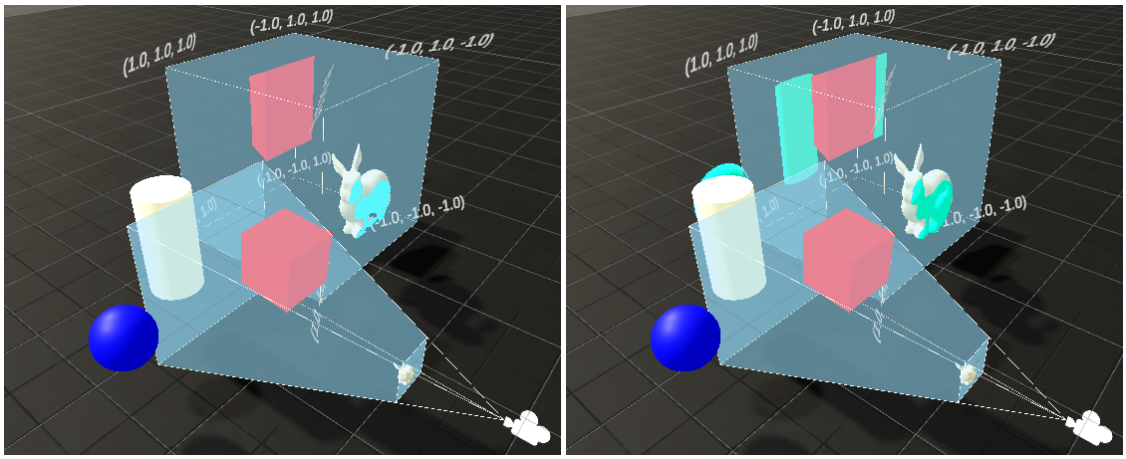
Rendered 2D image Looking in front of the *canonical view volume* [1], shows the resulted transformed scene. The direct comparison of this to a real rendered result (2D image) may make the progress clearer to the learner, because they sees the image covering the unit cube. To achieve this, a Unity *camera* game object was placed in the frustum's origin. A texture was provided as render target to get the rendered perspective of the camera on it. The actual values of the frustum are always provided to this camera. The size of this preview exactly fits for illustration with the view volume's unit cube and the preview image is placed in front of this cube. To make it possible to look behind this image plane directly into the front of the unit cube, the learner can make the image plane transparent by pointing on it. Figure 5.8 shows both states of figure 5.7 from the perspective of the camera as 2D image in default and transparent.

a preview by Unity was placed for comparison

```
1  // File PerspectiveTransformation.cs
2  public override Vector3 TransformationFunction(Matrix4x4 P, Vector4 v) {
3    // (1) Transform vertex to world space by TRS matrix M (already in eye space)
4    var vModel = M * v;
5    // (2) Transform vertex into clip space
6    var vPerspective = (P * vModel);
7    // (3) Normalize vertex for Normalized Device Coordinates (NDC)
8    var vNDC = vPerspective / vPerspective.w;
9    // (4) Transform vertex back to local space
10   return Mi * vNDC;
11 }
```

Listing 5.1: Vertex perspective transformation via Unity script

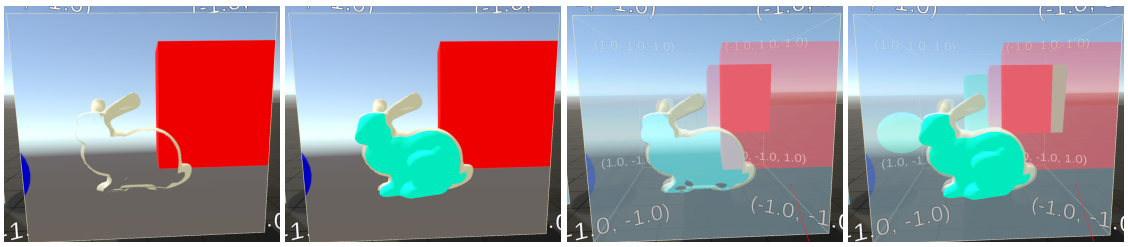
[11] Coroutine: `https://docs.unity3d.com/Manual/Coroutines.html` (visited on 10/20/2020)
[12] Unity Surface Shader: `https://docs.unity3d.com/Manual/SL-SurfaceShaders.html` (visited on 10/21/2020)



(a) Clipping with highlighted boundary

(b) Tinting instead of clipping

Figure 5.7: Realization of clipping



(a) Normal

(b) Overlapped tinted clipped geometry

(c) Normal (transparent)

(d) Overlapped tinted clipped geometry (transparent)

Figure 5.8: Render result as 2D image plane in VR

experiencing 3D Transformation and Clipping is possible

Implementation result Figure 5.5(a) shows the auto-generated frustum mesh, based on passed frustum values (z value for fear plane, z value for near plane, aspect ratio and FOV). The frustum mesh has a blue-transparent color and visualized wires (similar to those in *Triangle Builder*, section 5.3.2) to exactly see frustum's edges. The frustum representation can be changed by "dragging" its far and near planes across the z axis of the frustum or "dragging" sliders for FOV and aspect ratio. The "drag mode" is activated by pointing to the planes and pressing the trigger button on the controller and stops on releasing it. A scene will be transformed into normalized device coordinates, wrapped by a auto-generated canonical unit cube, with same blue-transparent material. The learner can observe the changes in real-time and explore the 3D transformations. Furthermore, they can toggle between clipping or tinting the clipped primitives and see the rendered end result in front of the unit cube on a plane. The rendered image is in front of the unit cube and fits exactly to the geometrical representation inside the cube. A test scene was created with a model of *Stanford bunny*[13] in the front, followed by a red cube, blue sphere and white cylinder. As the scene is only holding game objects with

[13] Stanford Bunny: `https://graphics.stanford.edu/~mdfisher/Data/Meshes/bunny.obj` (visited on 10/20/2020)

3D meshes, they can be easily exchanged for other purposes. Labels were placed on important coordinates and to name important objects (far plane, near plane, 2D image, etc.).

5.3.4 Scene: Guided Tour

This section describes the implementation of a Guided Tour itself and explains the design decisions. The Guided Tour is designed as a combination of the VW classes *exposition* and *experimental* (section 2.1.2): the world guides the learner through the scene, but let them experiment on selected parts.

Guided Tour needs a tour host

Tour host To create a Guided Tour, a "host" is needed [46]. A "tour host" conveys the story to the participants. To identify the host easily, its visual appearance should be distinctive, e.g. artificial hosts could be robots. Designing the best appearance of humanoid robots is a challenge [75] and the uncanny valley effect [76] plays an important role in the choice. To make an humanoid robot as "tour host", a "free to use" character asset of a humanoid robot *Space Robot Kyle*[14] (see figure 5.9) was used with a simple "idle" animation[15].

speaks to learner via text dialogue

Tour host dialog The tour host guides the participant through the tour. This is mainly done by printing the text the host is speaking. For this purpose the, already in section 5.3.1 mentioned, panel was used. Buttons can be enabled and disabled on it and the text can be changed to sequence the text Kyle is speaking. To follow user's position the component *Rotate To Transform* was implemented. The dialog is attached to the tour host and showing the front side in direction to the learner. Adding the *Rotate To Transform* script to Robot Kyle also lets him rotate to the learner. This creates a illusion to following learner's movements and always enables access to the dialog panel.

Unity Timeline was used for tour direction

Tour direction The Guided Tour consists of some sequenced guidance by the tour host and lets the learner do some interactions to directly "try out" the learned content. For the sequenced guidance Unity's Timeline is feasible with a combination of some organizational methods. Each guide chapter has an own tour director component (component *Playable director*). Every director and some organizer game objects get a *Signal Receiver* component to react on fired signals of a sequence. Each signal makes it possible to target game objects and call their methods. Figure 5.10 shows the tour director for *3D Transformation Stage*. The signals can be seen on the right as gray markers. The *Default Playables*[16] package completes Unity's timeline with useful Timeline assets. For example it has *TextSwitcher*. This timeline asset changes the text of a *Text mesh* component in Unity, which is useful for the previous mentioned dialog panel. Every tour director is paused each time the user has to finish a certain interaction. For example, the tour host is explaining a mechanism and only continues after the user has successfully executed this mechanism.

[14] Space Robot Kyle: `https://assetstore.unity.com/packages/3d/characters/robots/space-robot-kyle-4696` (visited on 10/25/2020)
[15] Idle character animation in Tobii Demo Scenes
[16] Default Playables: `https://assetstore.unity.com/packages/essentials/default-playables-95266` (visited on 10/25/2020)

constsis of a tutorial and 9 small stages

Guided Tour design As the persona *pupil* has less VR experience, the tour starts with a tutorial to become familiar with the environment and interaction. There Kyle explains pointables and interactables objects and lets the learner point on one. Next, the controller buttons are explained and finally the environment navigation by *teleportation pattern*. After completing the tutorial, a menu with stages of the *Simplified Rendering Pipeline* model appears (preview in figure 5.11(a)) with an additional button to hop into application stage. The learner can now select between these 9 small stages to get more explanations. To keep the learner in strict order, they can only select the next stage or repeat the previous stages. Every stage starts an own Unity Timeline *Playable Director* component instance to regulate the stage explanations and animations. All animations of a stage disappear, when a new stage starts. All stages are summarized in following:

1. **Application**: Kyle explains translation, scale, rotation and how they get to a "TRS matrix" together. For preview, each matrix is illustrated live above a red cube and each value of an example cube is changed step by step (figure 5.11(c)).

2. **3D Geometry**: Kyle explains the terms "vertices", "indices", "triangles" and "mesh". Further, he lets the learner build a first triangle and first mesh with previously described triangle interaction modes.

3. **3D Transformation**: Kyle explains the purpose of a frustum-based projection. A frustum illustration appears on the left (like in figure 5.5(a)) and the learner has to adjust interactively the far and near plane and afterwards the aspect and FOV (like in figure 5.5(b)). Kyle is waiting for learner until they sets the expected values.

4. **Clipping**: Here the tour host describes the clipping mechanism of the RP and hints to the toggle button for the clipping highlight, which I mentioned previously. The frustum and canonical view volume appears, including the frustum panel, to give the learner the opportunity to change the frustum and observe its clipping effect (see figure 5.7).

5. **Rasterization**: Kyle explains that fragments are potential pixels and they will be detected in this stage. Further he provides a sketch, inspired by Computer Graphics Lecture by Leif Kobbelt (figure 5.11(b)) instead of a great visualization.

6. **(Local) Lighting**: Kyle makes a brief mention that local light calculations are done in this stage. Instead of detailed explanations, a red sphere appears next to Kyle and the whole light for the user's VR scene is switched on and off in a one second interval. To not lose a user's orientation in the dark scene, the only highlighted objects are Kyle and the menu.

7. **Texturing**: Kyle briefly describes the purpose of a UV map and shows himself without texture and additionally blends in his texture on a panel (preview in figure 5.11(d)). On this panel the UV layout of his mesh is driving forward and backward to visualize the mapping.

8. **Visibility Test**: Because no sensible animations for z-buffer were implemented, this step is purely textual. Kyle explains that the z-buffer is used to store the nearest z values of all potential pixels to finally detect the pixels of the 2D image.

9. **Image**: The frustum and unit cube appear (like in figure 5.7), with addition of a 2D image preview (figure 5.8) of the result in front of the unit cube. Kyle invites the user to go in front of the unit cube, play around with the frustum and observe the results.

learner is able to experience a Guided Tour

Implementation result For the Guided Tour a tour host was placed. His name is Kyle and he is an animated humanoid robot. Everything he is saying, will be presented to the user via text. To keep the focus on the learner, Kyle always gazes with his body to learner's direction. The user has to trigger the "English" (or "Deutsch") (figure 5.9) button via pointer to start the tour in their preferred language. At first, the user has to follow instructions to become familiarized with VR controls. And after the tutorial, a menu appears (figure 2.6), where the user can get into each step of the Rendering Pipeline. Every stage has explanations, animations and sometimes interactions (details were explained before).

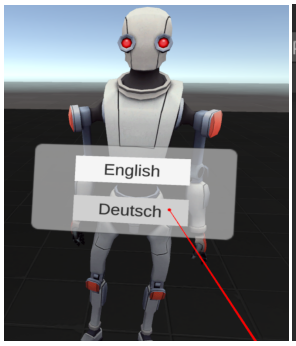


Figure 5.9: Space Robot Kyle and dialog panel

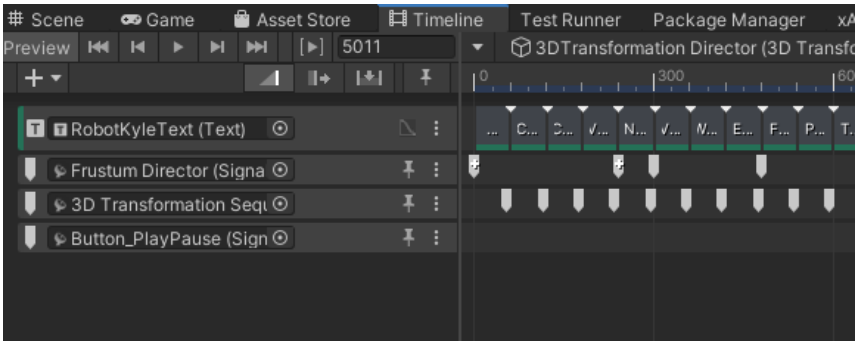
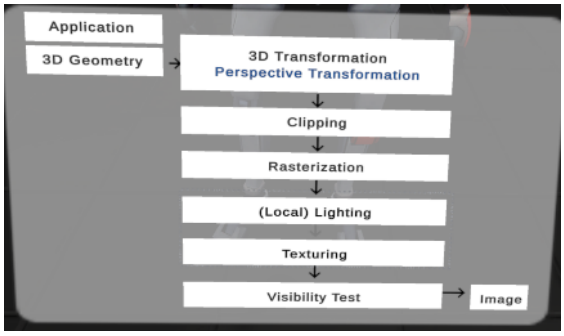
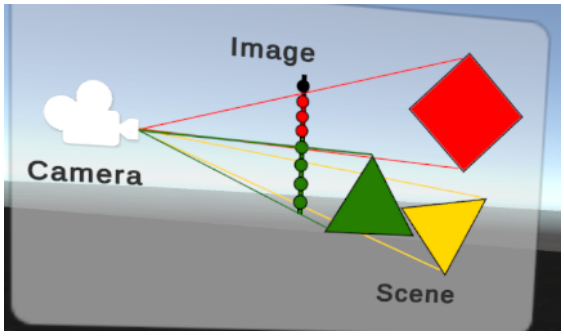


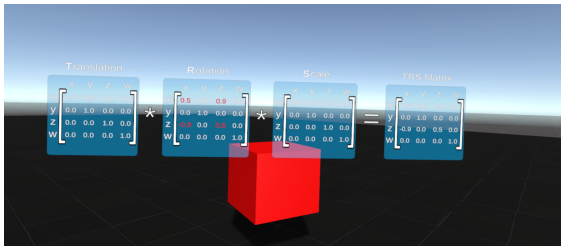
Figure 5.10: Tour director and signals



(a) Simplified Rendering Pipeline stage panel menu with prepended stage Application



(b) Rasterization image just projected on a panel



(c) Preview of transformation matrix composition. From left to the right: Translation, Rotation and Scale multiplied into TRS matrix



(d) Texturing preview. From left to the right: Kyle's texture on a panel with UV layout, Tour host kyle and menu, Kyle without texture texture

Figure 5.11: Guided Tour visualizations

5.4 Extending: LA Data Collector for VR

As already mentioned in section 4.3, I used the *Learning Analytics Data Collector for VR* [67] to support data collection of learner's breadcrumbs. Annabell Brocker created, in context of her approach, a web socket server, which connects the web panel and an Unity client to manage a xAPI tracking session. While integrating this system into my work, it made sense to extend it with more functionality. This section will describe the realized updates of the data collector.

5.4.1 Web Socket Server

Socket.IO was integrated

A. Brocker has developed a web socket server in NodeJS. There she coordinated a live communication between one Unity and one browser client. All events were handled by checking JSON properties via *if*-statements (example in listing C.3). Considering maintainability, adding more handler functions would decrease the overview. In Socket.IO events can be received and emitted with the function *on* and *emit*. To avoid a lot of "socket.on()" calls, a strict structure for event handlers was defined and following workflow was implemented: by adding new **.handler.js* files into the folder *handlers* (in figure D.3(b)), they will be automatically considered for event listening. Every in *[handlerName].handler.js* written function will be automatically used as Socket.IO event with prefix *handlerName*. For example: the function ":scenario" inside of *clients.handler.js* will be triggered, if the socket server receives the event "clients:scenario". Each function gets access to the socket instance via the variable *socket* and Socket.IO main instance via the variable *io*. In listing 5.2 are, for example, the events "clients:scenario" and "clients:calibration:start". The web socket body content is directly provided as function arguments and some helper functions distinguish the client type between *participant* (Unity) and *observer* (panel) clients. In the examples (listing 5.2), the function ":scenario" sends the Unity (participants) scenario information to observer clients and ":calibration:start" invokes calibration process on Unity clients. This kind of implementation offers to easily create new socket events with new files and functions. All created events can be seen in table B.3.

the socket server distinguish between observer and participant clients

```
1  // File clients.handler.js
2  ...
3  // clients:scenario
4  async ":scenario"({ name, gameObjects, actions, gestures }) {
5    const observers = getObserverSockets(io);
6    socket.scenario = { name, gameObjects, actions, gestures };
7    observers.forEach(b => b.emit("clients:scenario", {
8      ...socket.scenario,
9      id: socket.id
10   }));
11 },
12 // clients:calibration:start
13 async ":calibration:start"({ ids }) {
14   const participants = getParticipantSocketsByIds(io, ids);
15   participants.forEach(s => s.emit("clients:calibration:start"));
16 },
17 ...
```

Listing 5.2: Code snippet of new socket handling

Template Format In A. Brocker's solution [67] she used, so called, *templates* to save configurations of a tracking session. These templates were saved in *CSV (Comma-seperated-values)* file format. A template holds its template *name*, an *username*, *email*, *LRS address* and *URI for LRS*, *LRS Authentification Key* and *Secret*, selected *game objects* and *actions*. But this file format has some problems: CSV does not support nesting nor arrays. And as the reading or saving process needs strict order, it is difficult to extend the template settings. A pragmatic decision is to use JSON-format instead of CSV. With JSON it is possible to easily add new properties, without breaking any old data. And as the system is programmed with JavaScript anyway, the usage of JSON is more intuitive. Listing 5.3 shows the old properties in CSV format, where *gameObjects* and *actions* were concatenated with comma (e.g. "Sphere,Cube,Plane,...") and *lrs* and *uri* are URLs. The more game objects (or actions) were used, the longer one line gets and the file becomes less readable. In listing 5.4 is the new template format. Here, only types where used to define the new structure. The root is an array, to make it possible to hold multiple templates. Every template has the properties *_id, name, identity, auth, lrs, uri, scenario* and *tracking*. The old *setupName* was replaced by *name*. The properties *username* and *email* were grouped to *identity*. Further, *authUsername* and *authKey* were grouped together to *auth* and *gameObjects* and *actions* to *tracking*. To make a template editable, it was sensible to not only store active tracked information, but also storing information about the scenario itself. The property *scenario* holds the scenario's name and all its game objects and actions. To store which VR interactions has to be tracked, it was sensible to add the property *gestures* to *tracking* and *scenario* (all possible gestures).

Changed template format from CSV to JSON

JSON is more intuitive and flexible than CSV

Some properties can be grouped, Added scenario information and gestures

```
setupName;username;email;lrs;uri;authUsername;authKey;gameObjects;actions
```

Listing 5.3: Old templates CSV columns

```
[
  {
    _id: String,
    name: String,
    identity: { name: String, email: String },
    auth: { key: String, secret: String },
    lrs: String,
    uri: String,
    scenario: {
      gameObjects: [String],
      actions: [String],
      gestures: [String],
      name: String
    },
    tracking: {
      gameObjects: [String],
      actions: [String],
      gestures: [String]
    }
  }
]
```

Listing 5.4: New template format in JSON

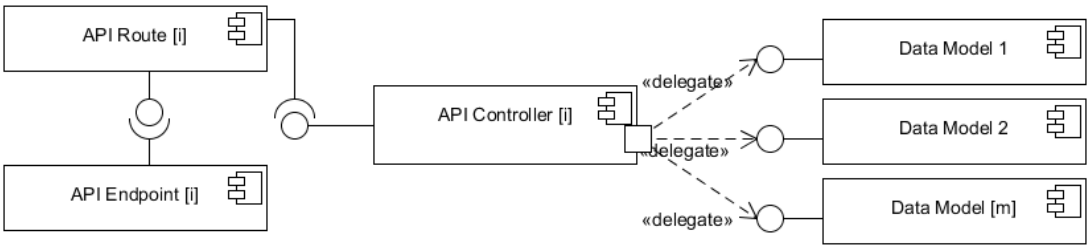


Figure 5.12: Abstract API data model

Storage methods: (1) server disk, (2) database, (3) import and export

Template Management In A. Brockers work [67] it was only possible to store new templates and load them again, instead of editing old. To make old templates update-able, it was necessary to identify every template by an unique id (property *_id* in listing 5.4). For the generation of an unique ID, the Node package *UUIDv4*[17] was used. There were three possibilities to load and save templates: (1) use server's local storage, (2) use a database instance or (3) let the user save and load template files from his/her own hard drive. After talking with a persona *researcher*, we have decided to use solution (3) at first and extend it later. This was solved by a simple import and export functionality via own *Vue* component, which will be discussed later in frontend implementation section. But to support a well ground for future extension of solution (1) and (2) a simple REST endpoint on the server application was created. The node server application does not only startup a socket server, it is also starts a HTTP server.
The API folder structure can be seen in figure D.3(a) and works as follows: all files in folder *routes* with the ending **.route.js* will be provided to *Fastify* with its *Fastify Routes*[18] definition standard. This will automatically create the defined routes. The file *templates.route.js* creates, for example, the default CRUD (Create Retreive Update Delete) endpoints in table B.4. All endpoints consume and provide the defined format of listing 5.4. A *route* file includes functions which are provided by controllers. The file *templates.controller.js* holds the functions *createOne, getOne, getAll* and *deleteOne* which use *Template.model.js* file to manage the resource *Template*.
Finally, the API system structure can be abstracted to the component model in figure 5.12: one endpoint is bound to one route, a route is using API controller, which is using multiple data models. This structure is easily extendible: to add a new resource, it is only needed to define three new files.
As the requirements for researcher will be clarified on later iterations (after this work), the researcher panel was kept to be thin implemented and the question of the most suitable database will not be discussed in this work. Instead, the current implementation simply uses a "local database"[19], to ensure a first well ground of the server instance.

Implementation result The web socket server is a NodeJS project and can be started with the command "node index.js". The node instance creates a web socket server for live communication purposes and a HTTP endpoint for template management. As it is a NodeJS implementation it can be easily delivered to a remote server (e.g. as Docker container).

[17] UUIDv4: `https://www.npmjs.com/package/uuidv4` (visited on 10/03/2020)
[18] Fastify Routes: `https://www.fastify.io/docs/latest/Routes/` (visited on 04/10/2020)
[19] node-localdb: `https://www.npmjs.com/package/node-localdb` (visited on 05/10/2020)

5.4.2 Researcher's Configuration Panel

The frontend for tracking configurations (figure D.5) in A. Brockers work was pretty simple and has filled her initial needs well. With her frontend version, it was possible to select current game objects and actions for tracking, setup LRS data and store the configurations as templates. The decision to upgrade her solution from web sockets to *Socket.IO*, made it necessary to update the web socket handling of the frontend, too. As her existing frontend was very simple and the configuration templates were extended by new attributes, it was offered to upgrade also the frontend and its functionality, too. To achieve a fast "working" solution, I have chosen my personal preference Vue in combination with Nuxt.js and Vuetify. The small re-design was oriented on A. Brocker's UI mockups [67], with some own ideas. Figure 5.13 shows relevant re-implemented classes of the frontend: The application has two main Vue components, which are used as Nuxt.js pages. A LRS client can be defined by authentication *host*, *secret* and *key*. This way it is possible to visit the researcher's configuration panel via analog URL query parameters. This panel is realized as *ClientConfigurationBoard* component. In general, the dashboard is composed by some Vue components. All components use helper classes to manage the data layer. These helper classes are shown in figure D.10 and don't need detailed explanation: all classes are ensuring a strict data structure for communication between Unity client and configuration panel. The components *RemoteTemplates* and *ImportExportTemplates* realize the template management on frontend's side and they have the same component inputs and similar look. The only difference is the functionality: *RemoteTemplates* uses the template API and *ImportExportTemplates* just uses system file browser to import them and offers a download method for export. To switch between two components the application can be toggled between "local" (for import/export) and "remote" (for API) mode via the environment variable *DATABASE_TYPE*. A default LRS url (*DEFAULT_LRS_URL*) and api url (*API_URL*) can be also configured through environmental variables.

it was offered to refactor the solution

refactoring was done with Vue, Nuxt.js and Vuetify

Implementation result Vuetify elements already provide easy responsiveness support. The startpage has two fields: *LRS Endpoint* and *Authorization (Header)* (figure 5.14). Here the researcher shall define their target *LRS url* and enter its *Basic Auth Authorization* string (all values can be picked from Learning Locker *Settings* > *Clients*). After clicking on the button "Get LRS Clients", all accessible LRS clients will be displayed (red area in figure 5.14)[20]. After choosing a client via button "Select LRS Client", the researcher is navigated to the configuration panel. The url holds all information[21]. This way, selected LRS clients can be added as bookmarks in a browser. The configuration panel can be divided into four functionality areas (figure 5.15):

researcher can manage multiple participant clients

1. **Selected LRS client card:** If any rights configurations inside of Learning Locker are not enough (e.g. recording xAPI statements are disallowed), then this part notifies the researcher. The other parts are disabled. The "Open Learning Locker" button opens the correct Learning Locker dashboard for an organization automatically.

2. **Loading and saving area:** This area is for the overall management of templates. They can be loaded or saved. As in figure 5.15 the panel is running in "local" mode, this area has "Import" and "Export" buttons. Templates can be loaded/saved from/on own computer as JSON files. In "remote" mode this area looks the same except for missing import/export buttons. Saving und loading requests are made automatically by just using "Save"

[20] All secret keys are used as example are not valid after publication of this work
[21] Configuration page url path format: /clients/:clientId?key=:clientKey&secret=:clientSecret&host=:lrsHost

button and page load. Individual templates can be deleted with "Delete" button, duplicates can be created through "Copy" etc.

3. **Template configuration:** The detailed configuration settings are represented here. Template name, identity name, identity e-mail, scenario name, xAPI base url, tracking gestures, actions and game objects can be configured here. Instead of simple HTML select elements, the tracking objects use a self-extended Vuetify select, called "v-compact-select". This element consists of many check boxes and a (un)select all option (example in figure D.9).

4. **VR clients card:** This area consists of all connected VR clients depicted in a table. The input fields are analog to a template. At the end of a table entry are four unnamed buttons in style of icons: (Starting from left) "Save as template" (for saving a VR client configuration as template), "Start calibration" (to start SteamVR eye calibration session), "Start/Stop tracking" (to start/stop tracking session) and "Quit application" (to close a VR application). Above the table are the same operations in style of wider buttons and names. These buttons are for the application of many selected VR clients at once. Every client has a status circle. Green stands for "ready", red for a missing configuration (e.g. needs calibration[22]). A blinking circle stands for an active operation (e.g. calibration, tracking).

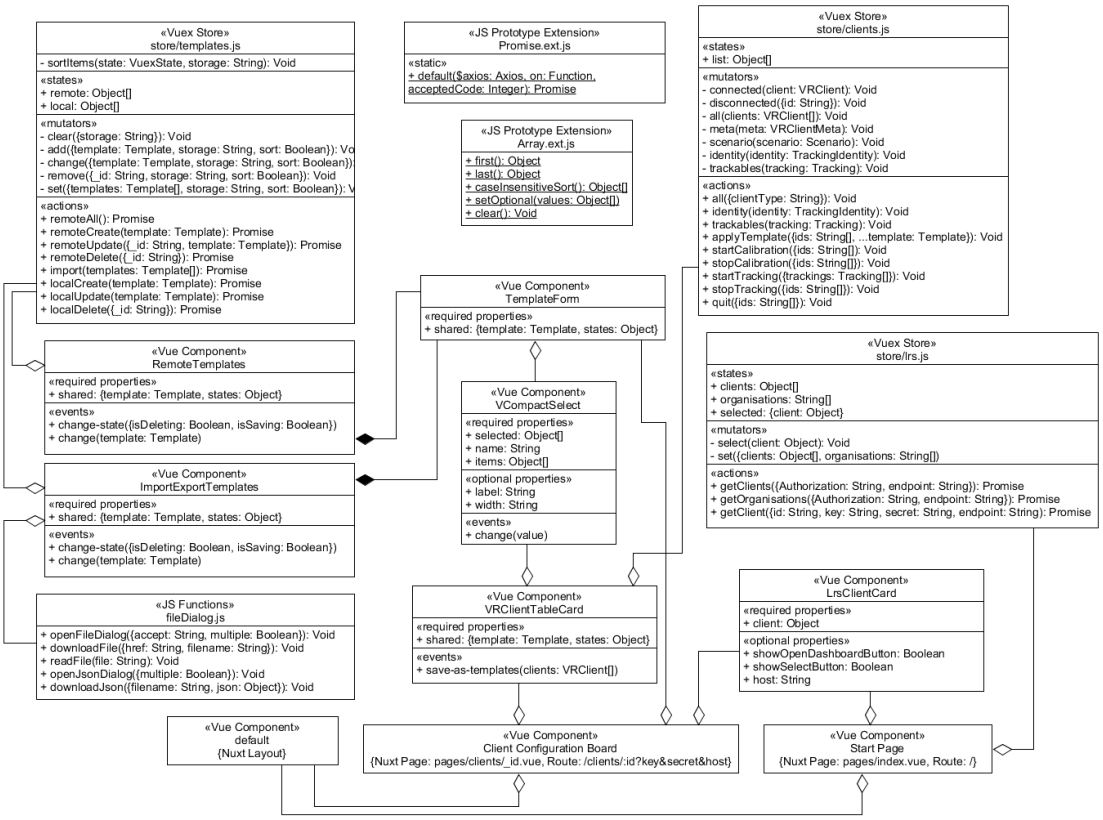


Figure 5.13: Frontend Nuxt.js Class diagram

[22] Status can be read with mouse over

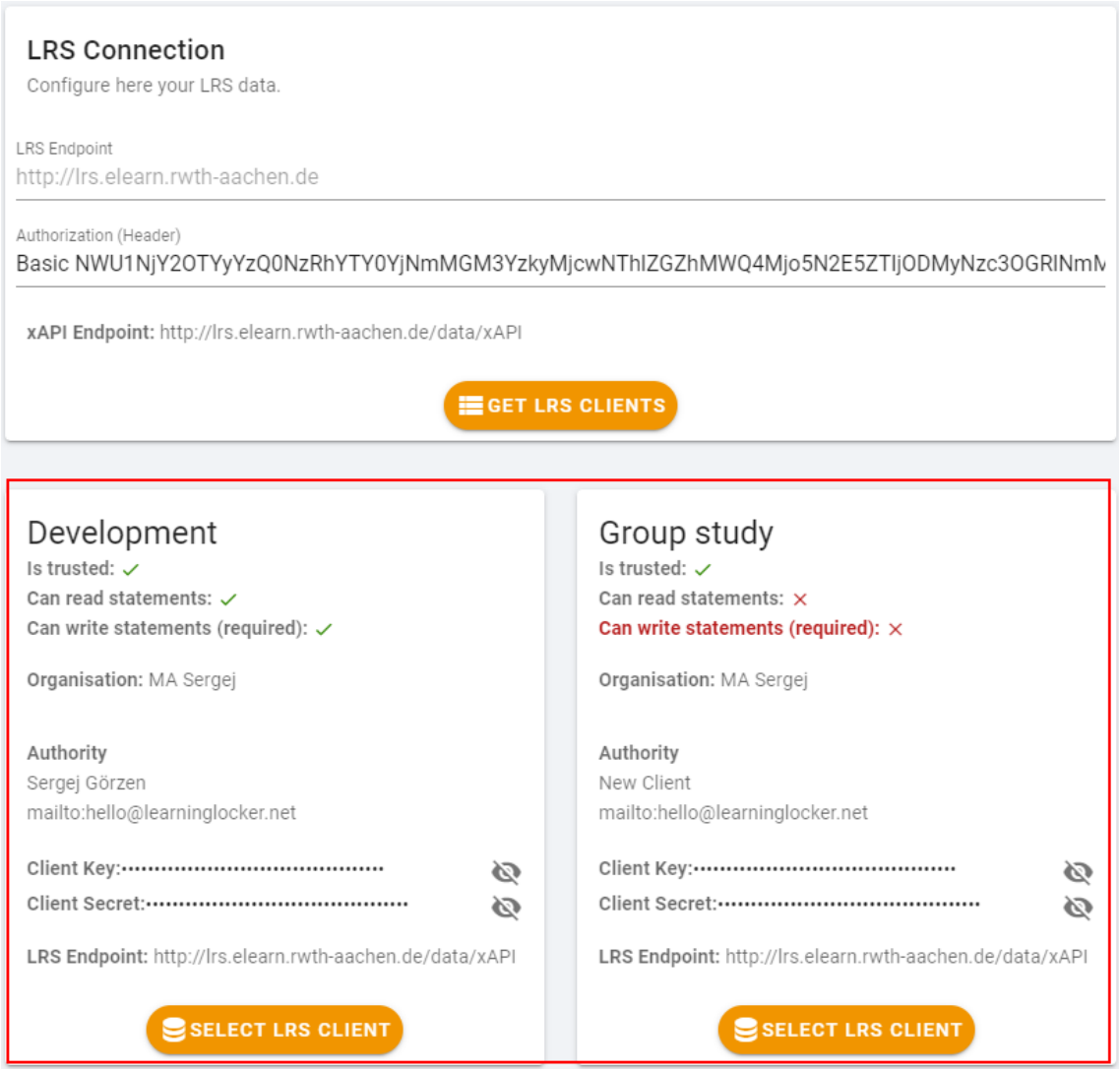


Figure 5.14: Researcher's configuration panel - Startpage

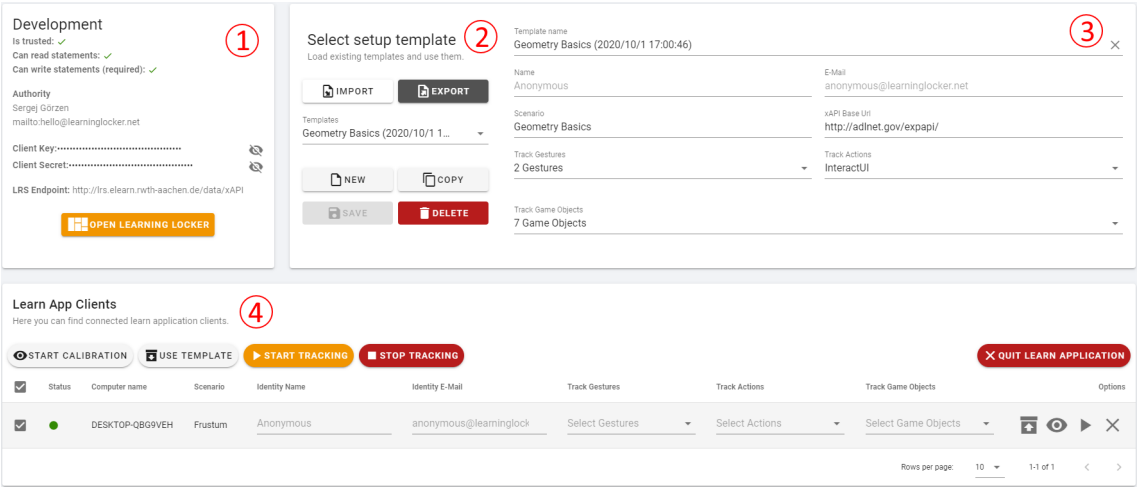


Figure 5.15: Researcher's configuration panel - Dashboard

5.4.3 VR Data Collector Unity

Regarding the requirements of the personas *researcher* and *developer*, it is necessary to support statement forwarding of xAPI statements. While developing learning scenarios, the developer wants to easily assign suitable statements to implemented learning tasks. This tasks will deliver data, the researcher can use to analyze the learning scenario. Providing xAPI statements inside of Unity needed a framework. A. Brocker had created in [67] first Unity scripts and prefabs to track eye focus and gaze of Unity game objects and to track controller actions. But this implementation had a big issue for my work: the data collector offered to track all "currently active" scene game objects. The first problem of this is issue is, that the researcher panel had offered also many irrelevant game objects (like "Directional Light" and "Camera") for selection. Another problem of the issue is, that new generated game objects had been ignored and cannot been considered for a tracking session. However, all VR scenarios of this work are very dynamic (e.g. transformed frustum objects, placed vertices). To make dynamic game objects trackable, this issue has to be resolved. Apart from some resolved issues, this library also became more structured and got better performance.

previous solution offered to track also unnecessary game objects and ignore new generated

Core elements The namespace *VRDataCollector.Core* was created for all old and new important structs and interfaces. The most important one is *IConsiderProvider*. Nearly every controller for xAPI statement (tracking controllers) should implement this interface. This way, it is possible to communicate a common method which should be used to provide tracking information.

important structs and interfaces

Tracking initialization The *Tracking Initializer* script of A. Brocker's work [67], was updated to the Socket.IO procotol (examples in listing 5.2) and received a feed with more "tracking controllers". This way, it is possible to decide which VR gestures should be considered (eye focus, eye fixation, head nodding, head shaking, pointing and interaction).

Tracking Initializer was extended

Tracking controllers The old version of the data collector had already started to use the modular component principle of Unity to create several controllers for own tasks (e.g. *ActionsController.cs*), but did not ensure it everywhere. To enforce this principle, a sub-namespace (*VRDataCollector.Controllers*) was created with some new added controllers (see table B.2). Every controller has its own scope and does not have knowledge about the existence of others (e.g. a *Teleport Controller* has no idea about *Head Controller*). The most important controller, in this section, is the *LRS Controller*. Each tracking controller requires this component to communicate with the LRS. More details about each controller can be found in table B.2.

more tracking controller were added

LRS Controller The *LRS Controller* does not only forward xAPI statements to a Learning Record Store, it also uses queues to keep failed statements in memory to send them again later. Before this refactoring, statements were automatically written to a local text file, immediately after failure. But, because of possible hard disk read/write delays, this step may be a bottleneck for real-time applications. A better solution is to load older (failed) statements from the file system on start and keep these statements in queues. When a connection to a LRS is established, queued statements will be sent to the LRS and failed statements will be queued again. To keep the failed statements for the next start, they will be saved in the file system when the application quits. But this approach does not improve real-time experience alone. To decouple the xAPI composition and forwarding from main game-loop, this progress was moved to

background manager for xAPI statements

background threads: two asynchronous background workers (one for queue management and one as "sender") were added. The LRS controller is, also holding, names of dynamic trackable game objects, to consider interaction with new generated game objects. The registration of game objects can be done by developers trough the method *LRSController.Register*.

game objects needs component Trackable to be considered for tracking

Trackables *Trackable* is the most influential data type for the whole tracking progress. A game object only occurs in researcher's configuration panel if it has this component. Boolean properties define whether the game object reacts to interaction, eye focus or other interaction kinds. Default interaction xAPI statements can be changed here, too. For example: for a VR chair the default *interacted* xAPI verb can easily be changed to *sit* without a lot of extra implementation.

Pointables react on learner's laser pointer and fire events

LaserPointer and Pointables A *Pointable* is a component, which can be used for reactions with laser pointer events. The laser pointer is based on *SteamVR_LaserPointer* and extends it with more reactivity. It fires events like "pointer inside of object", "pointer outside of object", etc.) and provide it to *Pointables*. This way *Pointables* can communicate with *Trackable* components and emit interaction statements. As the laser pointer uses Unity's *raycast*[23] to interact with game objects physics colliders, *Pointables* need a *collider* to be considered.

Implementation result While integrating the *VR Data Collector for Unity* by A. Brocker [67], the folder structure was refactored, the performance was improved, the maintainability increased and the consideration of dynamical generated game objects for tracking was enabled. Additionally, the pure web sockets implementation was replaced by Socket.IO protocol.

5.5 xAPI Definitions Fetcher

A. Brocker has baked xAPI statements into her *VR Data Collector* hard-coded on demand. This way is quicker, but has neither consistency nor redundancy reduction. Thus, I decided to realize a more consistent approach. As xAPI statements hold descriptions for their verbs or activities, it makes sense to define xAPI vocabulary in a list or database. There are several approaches to solve this: One possible solution is to load the vocabulary from a remote database. But this solution would create conflicts in different versions. As a consequence, an outdated verb could break the whole application. Another solution is to hold the vocabulary in a local database (SQLite, JSON, XML, ...) or in a hard-coded dictionary. These solutions have no problems with different versions and don't need Internet access during runtime. But they would not lead to a consensus in usage of the verbs, activities, extensions and their descriptions inside a working group (e.g. Learning Technologies Research Group). Moreover, it may cause an organizational overhead by merging definitions between multiple projects or adding them into a consistent database. Anyway, I contacted a research member of the Learning Technologies Research Group, who is representing the persona *researcher* and *developer* at the same time. He provided me some preferences for the usage of xAPI and recommended their own xAPI vocabulary with strictly defined xAPI verbs, activities and extensions. All are defined in JSON files (a example for the verb "interacted" in context *virtualReality* can be seen in listing C.2). This vocabulary lies on a GitLab repository[24] and makes a common consensus and a vision of TEL specialized usage. Additionally, GitLab does not only have a version control (and makes it

strictly defined xAPI instead of hard-coded statements

[23] Unity Raycast: `https://docs.unity3d.com/ScriptReference/Physics.Raycast.html` (visited on 10/14/2020)

[24] Learntech RWTH xAPI Repository: `https://gitlab.com/learntech-rwth/xapi/` (visited on 09/23/2020)

possible to checkout the newest or older versions), it is also creating a platform for discussing definitions and their descriptions.

5.5.1 xAPI Definitions

new contexts eyeTracking, gestures and virtualReality

As the repository had not enough definitions for this work, it was necessary to add some definitions. The repository already has the contexts: *generic*, *lms* (Learning Management System), *multitouch*, *seriousgames* and *systemControl*. These are not sufficient for a VR application. Thus, more contexts were added: *eyeTracking*, *gestures* and *virtualReality*. The hard-coded verbs and activities of LA Data Collector were packed together to the context *eyeTracking*. And obviously, a context *virtualReality* was created. But to strictly distinguish between physical actions and virtual actions, the creation of context *gestures* was needed. To be more concrete: *virtualReality* has xAPI components to describe interactions and targets in a VR scene and *gestures* focuses on the user's physical body interaction. And, of course, *gestures* can be used later for other kind of gesture detection (IR depth cameras, heart beat measurement, etc.). For this thesis, it was only used to transfer information about which hand was used and which hand is the primary one. Final results are listed in table B.5 (activities), table B.6 (verbs) and table B.7 (extensions).

5.5.2 Fetcher Tool

translated xAPI JSON definitions to C# classes

The definition of xAPI components, in the common repository of the Learning Technologies Research Group, ensured consistency and reduced redundancy. But the next challenge was to adding the grammar in programming code, without breaking consistency or redundancy. The usage of string key-value based hash tables or dictionaries would not resolve the problems: Typos of key names or not existing keys would break the functionality at runtime. Static C# classes and properties could help out here. But carrying xAPI definitions from repository into code by hand, would cause a big overhead of work and may create inconsistency again by doing typos. Thus, the final idea was to generate the C# classes from GitLab repository automatically. To achieve this, the tool *xAPI Definitions Fetcher* was created. This solution was packed together to a managed class library (DLL file) and a C# .NET Core 3.0 console application. The executable accepts arguments (or alternatively user inputs) as configuration and uses the DLL file classes to generate C# classes by fetching JSON based xAPI definition from repository. Figure 5.16 gives a brief overview of the whole procedure. The console application picks arguments (or user inputs) to know whether the fetcher should use a local or remote repository (argument *source*). A vocabulary source (e.g. *GitLabSource* or *LocalStorageSource*) implements the interface *IDefinitionSource* (a overview of all classes can be seen in figure D.11), which has a bunch of methods (like *GetActivities*, *GetVerbs* or *GetExtensions*). This ensures the easy extension of new sources in the future (for example FTP or GitHub). *GitLabSource* is using GitLab's API[25] to scan the repository file tree. After grouping JSON files as verbs, activities and extensions, the information about name and file location is stored in the struct *DefinitionMeta*. This information then helps to serialize JSON content into the struct *xAPI_Definition*. *System.CodeDom* was used to ensure a safe C# compilation, instead of pure string concatenations. *LocalStorage* only differs in the kind of fetching the JSON content: It is simply using *System.IO* namespace to scan directory file structures and read file contents. Finally, it ends with the serialization of JSON into *xAPI_Definition* structs, too. This whole generation is tailored for the GitLab repository

[25] GitLab API: `https://docs.gitlab.com/ee/api/` (visited on 10/11/2020)

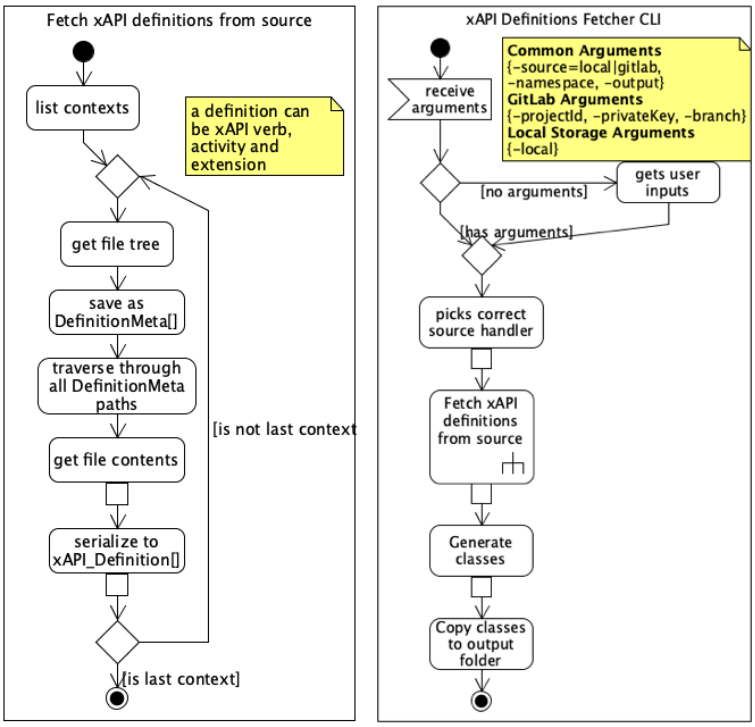


Figure 5.16: xAPI Definitions Fetcher Activity

of the Learning Technologies research group. The rough progress can be abstracted into the activity diagram in figure 5.16. An essential part of the code generation are the classes inside of namespace *Resources*. They officiate as base for the generated classes to ensure strict types and base methods. Inside of the fetching tool, they are only included as embedded resources and will be copied into destination folder.

C# CodeDom Generation On the left side of figure 5.17 is the file hierarchy of the xAPI definition repository. Each context has the folders *activities*, *extensions* and *verbs*. The extensions folder also has three sub-folders: *activity*, *context* and *result*. The right part of the figure represents the output result of the C# class generation with namespace *xAPI*. All auto-generated classes are based on the *Core* folder (which will be copied from fetcher's embedded resources). The core ensures strict types and basic methods. For example: every xAPI context will be an *xAPI_Context* instance, and will have all its class methods (*GetByName(string name)*, *GetVerb(string name)* and *GetActivity(string name)*). This makes it possible to extend functionality of all contexts by only extending the core class.

verbs, activities and extensions are collected in strict C# classes, extending some super classes

The folder *Definitions* holds all generated classes. Every context gets its own collection classes for verbs, activities and extensions. Every definition has its own instance of a dictionary for names and descriptions. Taking again the verb *interacted* (JSON from listing C.2) from context *virtualReality* as example: the generated class will be *xAPI_Verbs_VirtualReality* and the field entry has the key "interacted", names "en-US" and "de-DE" and it's defined descriptions. The exact code snippet can be seen in listing 5.5. The file class *xAPI_Context_VirtualReality* holds all such instances (snippet in listing 5.6). This is working analogous to activities. But extensions needed another design. The sub-classes *xAPI_Extensions_Activity*, *xAPI_Extensions_Context* and *xAPI_Extensions_Result* are deriving *xAPI_Extensions*. This base class is based on

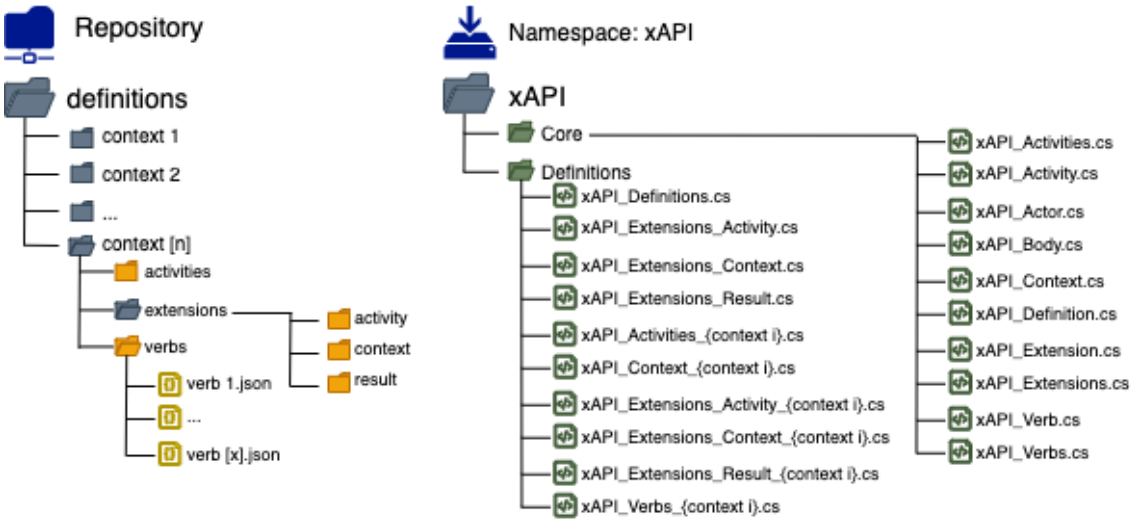


Figure 5.17: xAPI definitions file hierarchies overview

List<KeyValuePair<xAPI_Extension, object>>[26] to hold a list of *xAPI_Extension*s and on their assigned values (of any type). The method *Add* in *xAPI_Extensions* is used to chain many *xAPI_Extensions* instances together to one collection. Value assignment cannot be realized with simple *xAPI_Extension* instances. Instead methods, where a value can be passed, were generated. Those methods then return an extension instance, which then can be chained with other instances. The snippet of this implementation can be seen in listing 5.7. Finally all context instances will be collected in *xAPI_Definitions* and can be statically called from there (snippet in listing 5.8). The overall maximum amount of generated classes for n contexts is $6 \cdot n + 4$.

```
1  // File xAPI_Verbs_VirtualReality.cs
2  public class xAPI_Verbs_VirtualReality : xAPI_Verbs {
3    public xAPI_Verb interacted = new xAPI_Verb(
4      key: "interacted",
5      names: new Dictionary<string,string> {
6        ["en-US"] = "interacted",
7        ["de-DE"] = "interagierte" },
8      descriptions: new Dictionary<string,string> {
9        ["en-US"] = "An actor interacted with the VR object in the VR environment
             .",
10       ["de-DE"] = "Eine Akteurin interagierte einem VR Objekt in der VR
             Umgebung." });
11   ...
```

Listing 5.5: Generated verb interacted

[26] Alternatively *Dictionary<xAPI_Extension, object>*, but there was no need of hashed keys (less memory usage).

```
// File xAPI_Context_VirtualReality.cs
public class xAPI_Context_VirtualReality : xAPI_Context {
  public xAPI_Verbs_VirtualReality verbs = new xAPI_Verbs_VirtualReality();
  public xAPI_Activities_VirtualReality activities = new
      xAPI_Activities_VirtualReality();
  public xAPI_Context_VirtualReality_Extensions extensions = new
      xAPI_Context_VirtualReality_Extensions();
  public xAPI_Context_VirtualReality() : base("VirtualReality") {}
}
```

Listing 5.6: Generated context virtualReality

```
// File xAPI_Extensions_Context_VirtualReality.cs
public class xAPI_Extensions_Context_VirtualReality : xAPI_Extensions_Context {
  public virtual xAPI_Extensions_Context_VirtualReality vrObjectName(object
      value) {
    Add(new xAPI_Extension (
      extensionType: extensionType,
      key: "vrObjectName",
      names: new Dictionary<string,string> {
        ["en-US"] = "VR object name",
        ["de-DE"] = "VR Objekt Name" },
      descriptions: new Dictionary<string,string> {
        ["en-US"] = "Name of a VR object. Must be a string.",
        ["de-DE"] = "Namen vom VR Objekt. Muss ein String sein." }), value);
    return this;
  }
  ...
}
```

Listing 5.7: Generated context extensions for context virtualReality

```
// File xAPI_Definitions.cs
public class xAPI_Definitions {
  public static xAPI_Context_EyeTracking eyeTracking = new
      xAPI_Context_EyeTracking();
  public static xAPI_Context_Generic generic = new xAPI_Context_Generic();
  public static xAPI_Context_Gestures gestures = new xAPI_Context_Gestures();
  public static xAPI_Context_Lms lms = new xAPI_Context_Lms();
  public static xAPI_Context_Multimultitouch = new xAPI_Context_Multitouch();
  public static xAPI_Context_Seriousgames seriousgames = new
      xAPI_Context_Seriousgames();
  public static xAPI_Context_SystemControl systemControl = new
      xAPI_Context_SystemControl();
  public static xAPI_Context_VirtualReality virtualReality = new
      xAPI_Context_VirtualReality();
}
```

Listing 5.8: xAPI_Definitions Class

5.5.3 Strict xAPI Statements Showcase

As already mentioned in section 5.5, the member of the Learning Technologies Research Group I've contacted, informed me about their wishes for xAPI. He gave me feedback, from perspective of a developer, how the nesting of the generated C# classes shall be. The, in previous section explained, generation progress allows to use C# classes to bake xAPI statements, defined by the xAPI vocabulary of the Learning Technologies Research Group.
This section will show code snippets to show how the compose xAPI statements in C# with strict defined classes.

context selection

xAPI Definitions The main class *xAPI_Definitions* holds xAPI contexts and can be seen as the *root* of the generated collection. A small preview of auto completion for this static class can be seen in figure 5.18(a) (usage of listing 5.8). This offers to use every defined context.

decide between verbs, activities and extensions

Context As already explained before, every context has verbs, activities and extensions. The application of listing 5.6 can be seen in figure 5.18(b).

list verbs or activities as fields

Verbs and Activities Verbs and activities can directly be accessed by addressing them in a context class. Both are defined in C# like in listing 5.5 (activities use *xAPI_Activity* and *xAPI_Activities* instead of *xAPI_Verb* and *xAPI_Verbs*). The auto completion can be seen in figures 5.18(c) and 5.18(d). Finally, the verb *interacted* and the activity *vrObject* from context *virtualReality* can be used to create a xAPI statements (figure 5.19).

list extensions as methods to support values support of extension types activity, context and result

Extensions An extension can have an assigned value (e.g. VR object can have a name). This condition needed an extended scheme in class generation: using methods as representer of an extension, instead of fields, enables to provide a parameter. Figure 5.20 shows the creation of a statement with context extension *vrObjectName* and value "Red Cube" (at this point the generalized parameter *extensions* instead of *contextExtensions* would work, too).
As a xAPI statement can have multiple extensions, the *xAPI_Extensions* instance can hold multiple extensions. An example can be found in figure 5.21.

can call the fetcher via Unity

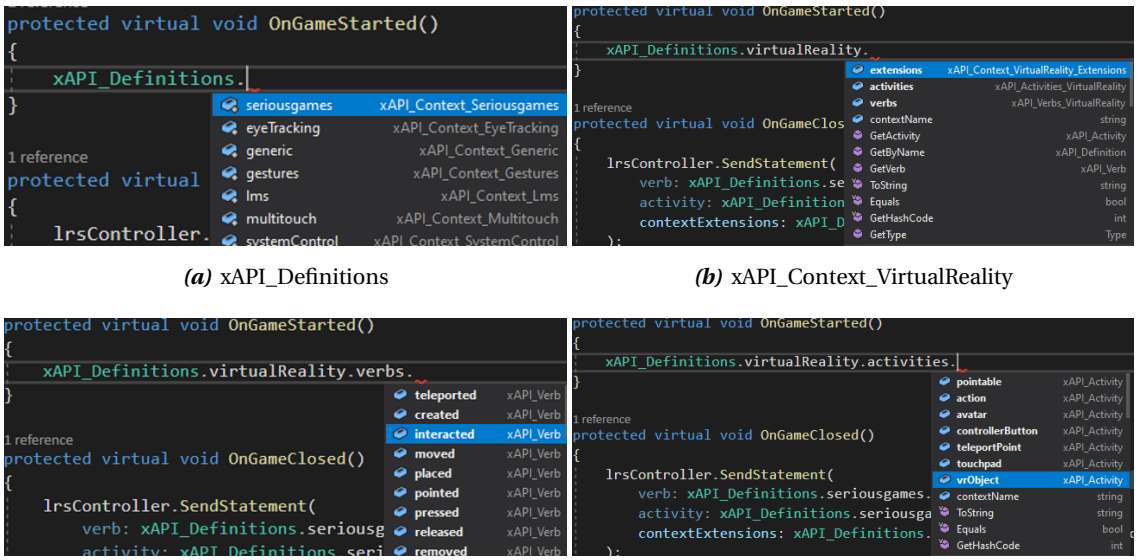
Unity plugin The xAPI definition classes in the previous section were transformed from JSON to C# classes. The fetching progress is started by an executable and can be configured via user input (figure D.8) or program arguments (e.g. as Windows Shortcut or C#'s Process call method). The more handy approach is an own Unity plugin, because the developer does not have to input the configurations every time. To get the Fetcher panel, it is important to have the *xAPI_Definitions_Fetcher_Window.cs* file (can be copied from fetcher's repository) to an "Editor" folder inside of Unity assets. Then, the fetcher can be opened by clicking on *Window* > *xAPI* > *Fetcher* of Unity's menu. All parameters can be configured via this Unity window. Figure 5.22 shows an example with all configurations. To run fetcher's executable the developer only needs to click on "Fetch xAPI Definitions".

Implementation result The persona *researcher* shall be have data about users' VR activities. From perspective of the research field Learning Analytics it makes sense to collect learners' activity breadcrumbs (section 2.1.1). This can be done with the help of xAPI standard (see section 4.2). But xAPI statements have to be programmed by developer. A common platform is the GitLab repository of the Learning Technologies Research Group. There, xAPI vocabulary is defined in JSON files (name, descriptions, etc.). GitLab offers version control and functions to discuss definitions.

researchers can define xAPI vocabulary in JSON

Previously the *xAPI Definitions Fetcher* was presented. This tool generates a xAPI vocabulary for C# from JSON definitions of a GitLab repository. This synchronization ensures consistency, reduces typos in source code and offers kind of strict defined library of xAPI definitions in C#. This solution supports to develop *EducationalVR* applications. The xAPI vocabulary can be fetched by the presented Unity tool and can be used to design xAPI statements for user tasks. A not ignorable benefit of this solution is that researchers can define their needed xAPI vocabulary and communicate to developers where they shall be used. Thus, this implementation supports the user stories of *researcher* and *developer*.

developer get this vocabulary directly as static C# classes



(a) xAPI_Definitions *(b)* xAPI_Context_VirtualReality

(c) xAPI_Verbs_VirtualReality *(d)* xAPI_Activities_VirtualReality

Figure 5.18: xAPI Definitions Usage Auto Completions

```
VRDataCollector.xAPI.CreateStatement(
    uri: "http://example.com",
    actor: new xAPI_Actor(name: "Max", email: "max.mustermann@example.com"),
    verb: xAPI_Definitions.virtualReality.verbs.interacted,
    activity: xAPI_Definitions.virtualReality.activities.vrObject
);
```

Figure 5.19: Create xAPI statement

```
VRDataCollector.xAPI.CreateStatement(
    uri: "http://example.com",
    actor: new xAPI_Actor(name: "Max", email: "max.mustermann@example.com"),
    verb: xAPI_Definitions.virtualReality.verbs.interacted,
    activity: xAPI_Definitions.virtualReality.activities.vrObject,
    activityExtensions: xAPI_Definitions.virtualReality.extensions.activity.vrObjectName("Red Cube")
);
```

Figure 5.20: Create xAPI Statement with context extension

```
VRDataCollector.xAPI.CreateStatement(
    uri: "http://example.com",
    actor: new xAPI_Actor(name: "Max", email: "max.mustermann@example.com"),
    verb: xAPI_Definitions.virtualReality.verbs.interacted,
    activity: xAPI_Definitions.virtualReality.activities.vrObject,
    extensions: new xAPI_Extensions
    {
        xAPI_Definitions.virtualReality.extensions.activity.vrObjectName("Red Cube"),
        xAPI_Definitions.generic.extensions.result.timeSpan(System.DateTime.Now)
    }
);
```

Figure 5.21: Create xAPI Statement with extensions

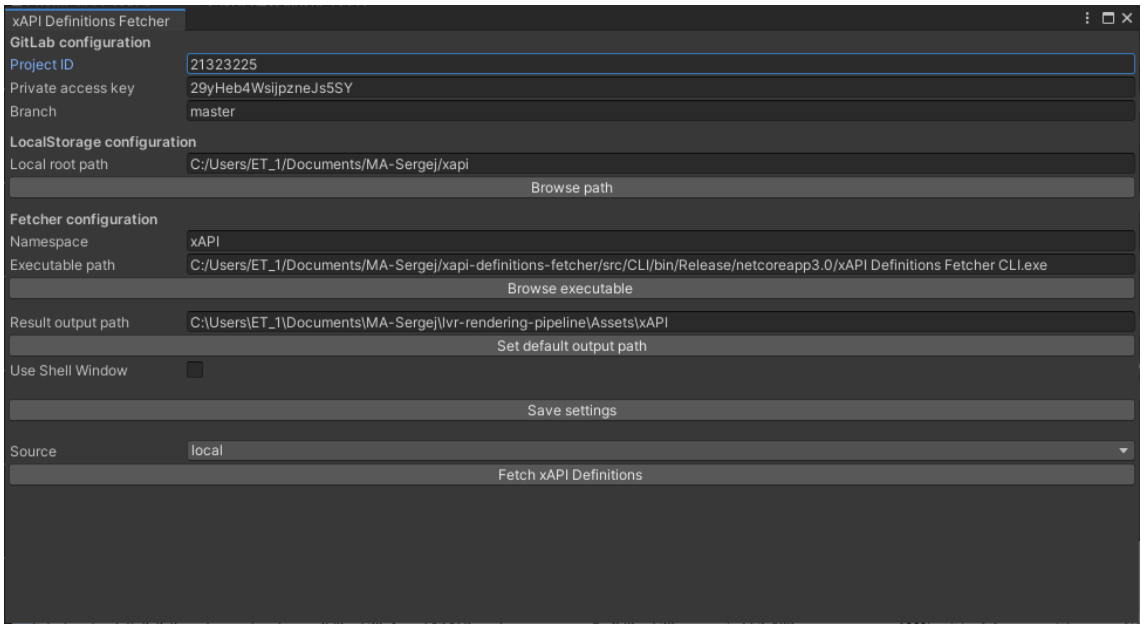


Figure 5.22: xAPI Definitions Fetcher Unity Window

5.6 Implemented xAPI Statements

Table B.2 lists some controllers for the regulation of specific xAPI statements (teleport, interaction, gaze, ...). It is still needed to get an overview of all xAPI statements, which are possible in context of the overall project. Table 5.1 summarizes the overall composed xAPI statements and the user actions to trigger them. This table consumes xAPI vocabulary from section 5.5 and uses following convention for the verbs, activities and extensions: verb/activity$^{\text{context}}$ and extension$^{\text{context}}_{\text{extension type}}$. A context can be *ET* (eyeTracking), *Gn* (generic), *Gs* (gestures), *SG* (seriousGames), *SC* (systemControl) and *VR* (virtualReality). And a extension type can be *ac* (activity), *ctx* (context) and *res* (result). For example: the first row of table 5.1 means the xAPI statement "Actor pressed action" is sent with the extensions "actionName" and "hand", when a controller button was pressed.

When triggered	Verb	Activity/Object	Extensions
Used controller button	pressed^{VR} | released^{VR}	action^{VR}	actionName$^{VR}_{ac}$, hand$^{Gs}_{ac}$
Pointer enters object	pointed^{VR}	vrObject^{VR}	vrObjectName$^{VR}_{ac}$
Press trigger while pointing on object	interacted^{VR}	vrObject^{VR}	vrObjectName$^{VR}_{ac}$
Timeline playable plays/pauses/stops/resumes	started^{SC} | paused^{SC} | ended^{SC} | resumed^{SC}	stage^{SC}	name$^{SC}_{ac}$
Visits/leave teleport point	entered^{Gn} | left^{Gn}	teleportPoint^{VR}	vrObjectName$^{VR}_{ac}$
Teleports in area	teleported^{VR}	player^{Gn}	position$^{SG}_{res}$
Changes Unity scene	teleported^{VR}	player^{Gn}	position$^{SG}_{res}$, level$^{SG}_{ctx}$
Opens/closes Unity	started^{SG} | ended^{SG}	game^{SG}	game$^{SG}_{ctx}$, gamemode$^{SG}_{ctx}$

Table 5.1: xAPI Composition Overview. Contexts: *ET* (eye tracking), *Gn* (generic), *Gs* (gestures), *SG* (serious games), *SC* (system control) and *VR* (Virtual Reality). Extension types: *ac* (activity), *ctx* (context) and *res* (result).

Chapter 6 Evaluation

This chapter delivers in the first section results about technical correctness of the implementations. The other section evaluates the conducted pilot study.

6.1 Technical Evaluation

According to Chopra [74] software tests increase the maintainability and there are infinite number of possible test cases. Test design can be a very complex. There are many types of testing and it is not sensible to apply them all. As the scope of this thesis is limited it makes to reduce the tests into the most necessary, which are ensuring the "technical correctness" of the implemented software. Software can be "verified" (human-based checking via inspections or walk throughs) and "validated" (computer-based execution of code to test "black-", "white-", "gray-boxed"). This work has following challenges for test cases: the VR application has 3D visualizations and interactions, the VR application and researchers configuration panel have to communicate with each other via a web socket server and the xAPI statements has to be composed correctly.

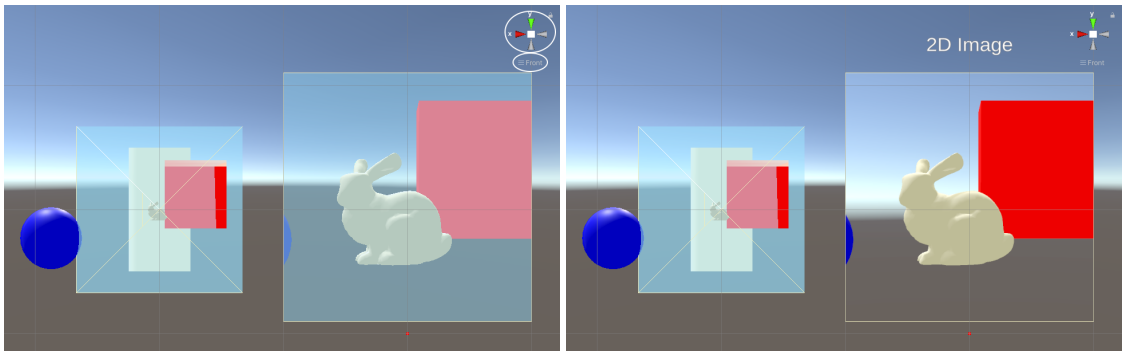
human-based vs. computer-based

This section represents the technical evaluation of the essential implementations. This is included to check the correctness of frustum transformations, xAPI statement composition and the client communication.

Perspective transformation and clipping of the demo scene

One possibility is to test calculations of the frustum and evaluate them. But as the transformation and clipping is a visualization it makes sense to "see" the result. Anyway, as the development should focus on the user experience, traditional ways of software measurements have to be given up for VR [43]. Another way to prove the correctness of the perspective transformation and clipping is to compare the transformed 3D scene with a, from Unity rendered, result. The 2D image shall looks the same, as "looking" in front of the canonical view volume. A visual comparison would "show" the correctness of the calculations. To do this, it is necessary to turn off perspective's distortion of Unity's default rendering progress. This can be achieved by switching to the isometric mode inside Unity's editor. The switch can be done by clicking under the Scene Gizmo (figure 6.1(a)). After activating the isometric mode, the perspective's vanishing points disappear and the Editor changes into a kind of "2D view" of the scene. Finally, it is possible to look in front of the unit cube by clicking on the Z axis on Gizmo, without any perspective's deception. Comparison between figure 6.1(a) (without 2D image) and figure 6.1(b) (with 2D image) shows the same result. As figure 6.1(b) is rendered by Unity's default camera component, then in fact it proves a correct perspective transformation and clipping of geometry for this scene in figure 6.1(a).

correctness can be verified visually



(a) Looking in front of Unit Cube ***(b)*** Overlaying 2D image in front of Unit Cube

Figure 6.1: Frustum Transformation Evaluation

Unit Testing

unit tests: small portions of tests

Unit tests check the correctness of a software [74]. How the name is already telling: unit tests are testing "units" (small parts) of the software. This can be, for example, methods or classes. In long-term unit tests are following a test plan and will create a great coverage of the system. Visualization- or interaction-based implementations (e.g. the frustum or the triangle building) cannot be tested with Unit tests easily. Thus, only sensible methods will be tested.
In further paragraphs all implemented unit tests will be described with the decision of their purpose.

test of event handling with mocha

Web Socket Server The web socket server has 10 essential events (table B.3). To ensure the correct handling of these events unit tests were created with *mocha*. The tests can be executed with command line "npm run test". A test run opens a Socket.IO server instance and connects two clients to it: *participant* and *observer*. The *participant* shall represent a Unity client and the *observer* the research panel client. Each unit test checks the correctness of its assigned event handler. For example: The event "clients:scenario" (listing 5.2) provides information about the active scenario of a participant to a observer client . The unit test for this function lets the *participant* send pre-defined scenario information to the server instance. The *observer* instance receives this information and compare them with the original data (example *mocha* code can be seen in listing 6.1). The window in figure 6.2 shows an example, where each test passes. If a handler is not working as expected, this window shall throw an error.

```
// File test.js
...
// Unit test for "clients:scenario"
describe("clients:scenario", function () {
  it("Observer should receive the correct scenario information of participants
      when the 'clients:scenario' event is emited.", function (done) {
    const testScenario = require("./test-scenario.json");
    observer.on("clients:all", clients => {
      const ids = clients.map(c => c.id);
      const participantId = ids[0];
      observer.on("clients:scenario", ({id, ...scenario}) => {
        expect(scenario).to.eql(testScenario);
        expect(id).to.equal(participantId);
        done();
      });
      participant.emit("clients:scenario", testScenario);
    });
    observer.emit("clients:all", {clientType: "participant"});
  });
});
...
```

Listing 6.1: Unit test of web socket event "clients:scenario"

```
Socket Server
  √ Connection
  [Observer] clients:init
    √ Observer should receive a clientType 'observer' when the 'clients:init' event is emited.
  [Participant] clients:init
    √ Participant should receive a clientType 'participant' when the 'clients:init' event is emited.
  [Observer] clients:all
    √ Observer should receive a collection of clientType 'participant' when the 'clients:all' event is emited.
  clients:quit
    √ Participant should be 'quit' when the 'clients:quit' event is emited.
  clients:setup
    √ Participant should be receive the correct template when the 'clients:setup' event is emited.
  clients:meta
    √ Observer should receive the correct meta of participants when the 'clients:meta' event is emited. (114ms)
  clients:scenario
    √ Observer should receive the correct scenario information of participants when the 'clients:scenario' event is emited. (115ms)
  clients:calibration:start
    √ Participant should react when the 'clients:calibration:start' event is emited.
  clients:calibration:stop
    √ Participant should react when the 'clients:calibration:stop' event is emited.
  clients:tracking:start
    √ Participant should react on correct tracking information when the 'clients:tracking:start' event is emited.
  clients:tracking:stop
    √ Participant should react when the 'clients:tracking:stop' event is emited.
```

Figure 6.2: Preview of running tests with *mocha* for web socket server.

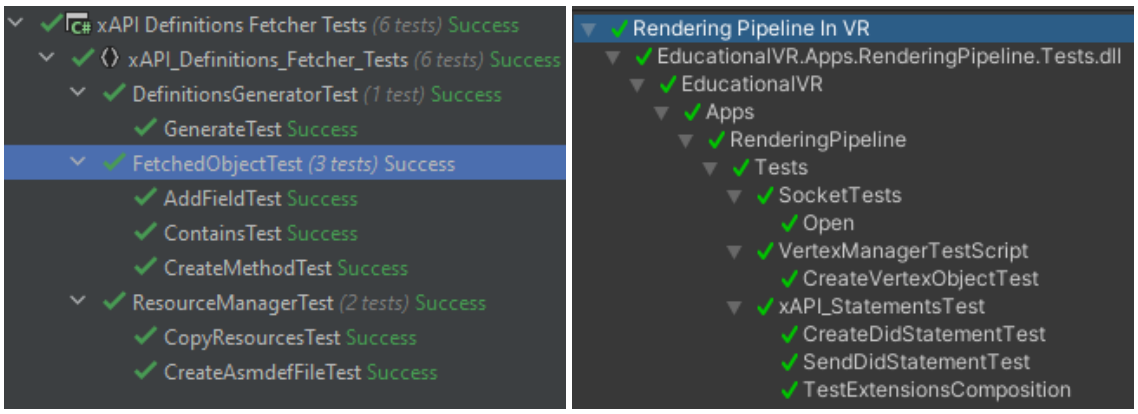
xAPI Fetching Changes of xAPI definitions in JSON files have to be updated by triggering a new fetch. As the C# language is strict, not existing assemblies (classes, fields or methods), will be noticed by the Unity editor or C# IDE. Thus, changes of the xAPI vocabulary inside the repository will be reflected in the C# classes, too. The *xAPI Definitions Fetcher* takes a storage *source* and generates C# classes based on the defined xAPI statements. To validate the generation with Unit tests, the most essential methods were identified and tested. The most important classes of the diagrams from figure D.11 and D.12 are *DefinitionsGenerator*, *FetchedObject* and *ResourceManager*. In one sentence: All classes will be generated by *DefinitionsGenerator* and

essential generation progress was tested, output was valided

sub-classes of *FetchedObject* based on the classes delivered from *ResourceManager*. A check of the essential procedures, here, would ensure correct fetching progress. For *FetchedObject* tests were created for those methods, which are responsible for the composition of fields (*AddField*) and methods (*CreateMethod*) of the generated classes. The *ResourceManager* was tested on its' only functions which copy the needed resources into target folder. Finally, the main generation progress was tested within the Unit test of the method *Generate*. To do this, a new *source* was created named "TestSource". This *source* has hard-coded xAPI definitions. This way, the only generation progress will be independently validated. In this test, the C# classes of *TestSource* will be generated. Afterwards, the content of those files files will be compared with pre-defined "correct" files. Generated files with wrong format would be recognized here. A preview of the unit test execution can be seen in figure 6.3(a).

xAPI Statement Composition and Forwarding From xAPI definitions generated C# classes can be used in C# code to compose xAPI statements. These statements can be forwarded to Learning Locker. Both procedures can be tested with unit tests. To achieve this, a *xAPI_Verb*, *xAPI_Activity* and three *xAPI_Extension* instances were created from hand in code. This ensures expectable results on their usage. Supposed the generated classes have a correct content (checked in previous paragraph), a validation of the hard-coded vocabulary would validate also the functionality of the automatic generated classes, because they are all following the same scheme. One unit test was created to check the creation of a *TinCan* statement. Another unit test is send a test statement to Learning Locker. The statement was saved successfully in Learning Locker, when the execution of *SendDidStatementTest* is successful. A preview of the stored verb can be seen in figure D.7. Additionally the connection to the web socket server is also tested via a unit test.
Together the connection to socket server, xAPI statement creation and the forwarding to Learning Locker are covered by these tests.

Correctness of statement creation and forwarding was tested



(a) Unit tests for xAPI Definitions Fetcher.

(b) Unit tests for Unity

Figure 6.3: Preview of running unit tests with *NUnit*

xAPI in Production

In previous section small portions of functionality tests were described. This section will do a xAPI validation in larger scale. One possibility to verify software is a human-based testing

method [74]. For example, the user can be guided through an instruction and do tasks. The implemented Guided Tour itself, is a exactly offering this kind of situation. Thus, a valid method could be to test the functionality by following the Guided Tour instructions. Beside testing usability and functionality, the xAPI statement transfer itself can be tested in larger scale. Previously, the transfer was tested for one statement per unit test run. This technical evaluation was combined with the pilot study. More details about the pilot study can be found in section 6.2 and the descriptions of the visualizations and interactions of the Guided Tour can be found in section 5.3.4.

Guided Tour can be itself the "walk through" of human-based tests

This section will discover the results in Learning Locker and check this way some implemented xAPI statements (table 5.6).
The three last test sessions collected overall 13530 xAPI statements on the LRS system Learning Locker. Even though the amount of test persons is not enough to make well-grounded predictions about the learning content from a didactic perspective, it is anyway possible to make some statements with only counting laser pointer interactions.
To focus only on the pointing pattern interactions of the Guided Tour scenario, following configurations were made on the updated researcher configuration panel (figure D.14):

- all 39 available game objects of the Guided Tour scenario were selected,
- the controller action *InteractUI* (by pressing controllers' *trigger* button) and *Teleport* (by pressing on trackpad) were chosen
- pointing gestures were counted.

Figure 6.4 shows a screenshot of a Learning Locker dashboard, for the collected data from pilot study. In the following are some interpretations of these charts:

1. Participant 1 and 2 were both more than twice active than participant 3.
2. The most with the pointer focused object was the "Simplified Model Panel" (figure 5.11(a)).
3. The "Triangle Builder feature" (building meshes) was much more used than the "Frustum feature" (3D transformation).
4. The last stage (*Image*) and the stage *3D Geometry* were visited the most. Both have the more complex interactions and visualizations. This can be an indicator for increased interest about interactions and animations, instead of the less designed stages.

As it was possible to configure the tracking session, start LRS recording (tracking) and to close in the end the Unity application over the researchers' configuration panel, the communication has been verified successfully. The previously stated interpretations can be confirmed by the recording from the digital appendix. In summary, the xAPI statement composition and forwarding has been proved on their correctness. As each participant of the pilot study has finished the Guided Tour, each implemented interaction and visualization passed the "walk through". Thus, they have been verified.



Figure 6.4: Dashboard with charts of the pilot study in Learning Locker

6.2 Pilot Study

According to section 2.2, an iterative progress is sensible for the development of VR applications. Conceptual requirements were analyzed and the content of the learning application was declared during supervisors feedback sessions. In order to not exceed the current scope, the conceptual content (chapter 3) was reduced to the final implemented results. A prototype with only essential features (like 3D transformation and clipping) has less unnecessary implementations and can be easier customized for later evaluated requirements. The first prototype tries to provide pupils an idea of the Rendering Pipeline. From design perspective, the pilot study iterations help to identify usability and design issues (like content difficulty). From research perspective, the qualitative feedback of experts can help to adjust the concept of an Educational VR Application and to extend xAPI specification. The results of this pilot study will be used in further research.

identify bugs, usablity and design issues

This section describes the study design and results.

6.2.1 Methods

The Corona pandemic caused required restrictions induced by the government. According to the restrictions for North Rhine-Westphalia[1], unnecessary contacts has to be avoided. To respect these restrictions, the study was conducted with overall eight participants on two places over a few days: one office at home and the laboratory of the Learning Technologies Research Group. In addition, the choice of test personas had to be restricted.

office at home and laboratory, 8 participants

[1] NRW Corona Restrictions: `https://web.archive.org/web/20201119221149/https://www.land.nrw/de/pressemitteilung/nordrhein-westfalen-setzt-vereinbarte-massnahmen-zur-bekaempfung-des-corona-virus` (visited on 11/19/2020)

Each test session took between 35 and 75 minutes. All participants were between 25 and 40 years old. They explored the Guided Tour of the Rendering Pipeline on their own. To get immediate feedback of their VR experience, we used the "think aloud" method with a followed free interview [77], [78].
To avoid the identification of repeating issues or bugs, already identified realizable issues were resolved immediately after each test session.

Home setup preparations Four of eight participants were invited to the home setup on different days. The room was ventilated well and the equipment was cleaned before each test session. The setup is a typical home workplace and the VR experience is limited to a $2x1.5$ meters room space. A picture of the home setup with one test person can be seen on figure 6.5(b).

four participants tested only the usability and interactions

The participants had no knowledge about Computer Science nor Computer Graphics. Thus, they had less or equal knowledge than the persona *pupil* and were good candidates for *worst case scenarios* (no pre-knowledge). As each of these participants also had no experience in VR, the equipment was explained at first. To avoid feeling of unsureness, they got briefed about that their goal shall be to not understand each detail. They shall concentrate on understanding the VR interactions and express their feelings of the experience.
This approach of not overburdening the user is consistent with other studies dealing with novices, see [79].

Laboratory preparations The laboratory tests were conducted within one week. To respect the Corona induced restrictions, the study was conducted remotely. My supervisor invited and briefed the participants, prepared the VR equipment in a $3x3.5$ meters laboratory, enabled me screen control access to the computer via the online meeting tool *Zoom*[2]. A picture of a remote test session can be seen in figure 6.5(a).

four participants have done expert tests and gave quality feedback

All four participants were experienced in user studies and had expert knowledge either in Computer Science or didactics. Each participant had covered another perspective and provided expert feedback:

- The first participant covered the perspective of *researcher*. This participant had explored the VR application and the configuration panel of section 5.4.2.
- The second participant is an expert in didactics and educational applications. This participant had explored the VR application from perspective of a lecturer (*teacher*) and provided feedback about didactic and usability aspects.
- The third participant had some pre-knowledge in Computer Graphics and covered the perspective of the persona *student*. This participant tried to understand the learning content of the VR learning scenario.
- The fourth participant had no knowledge about Computer Graphics but enough mathematical background to potentially understand the learning material. This participant represented the persona *pupil* and tried to understand the learning material.



(a) Picture of the laboratory via Zoom.

(b) Picture of home setup.

Figure 6.5: Pictures of the pilot study.

xAPI was only tracked for three participants

As the test sessions of the home setup and the first one of the laboratory focused only on interaction issues, xAPI statements were not tracked. We collected LRS records for the last three sessions, which where placed in the laboratory (more details about statistics of the recorded results can be found in section 6.1).

6.2.2 Results

noticeable observerations, important changes

As already mentioned, realizable feedback from participants was immediately integrated for next followed sessions. Resolved bugs will not be mentioned here. Instead, this section describes noticeable observations and provides important changes, which could be helpful for future Guided Tour or VR designs. Further ideas and hints for the future can be found in section 7.1.

the impressions were positive, no VR sickness

User impressions Each participant liked the VR experience and reacted positively on the humanoid robot as tour host. The experience itself was "exciting" and some of them said "it helps to understand things". More concrete: one of them noticed "light is providing color" and all of them understood that 3D geometry is based on triangles and the frustum defines the visible area. Especially the third participant (with CG pre-knowledge) mentioned that this application could help to understand the lecture material better. Five of the participants experimented with the frustum and found it "very interesting" to see what a very bright frustum does. Nearly everyone had shaped the frustum so, that all scene objects were visible and finally excluded the bunny 3D object from visible area.
Overall, no participant had signalized any symptoms of VR sickness.

different VR experience levels were noticeable

Different VR experience levels The different VR experience levels were noticeable: Participants with some VR experience physically moved around and tried out different controller buttons. The participants without any VR experience were very careful and unsure. They always pressed only the *trigger* button and didn't move physically.
This confirms the observations in [79].

autoplay of text impressed very negative ⇒use manual dialog control instead

Manual dialog control Unity's Timeline mechanism lets Kyle's dialog text run automatically. This made it difficult to choose common sensible durations for each text message the user could follow. The first participants missed some text messages and felt stressed because of it. On the other side, one participant repeatedly pressed the *trigger* button while pointing on the text and awaited to "skip" the dialogs. Thus, a manual control for dialog was created: each message takes only one second and is paused after completing. The users can decide themselves when they want to continue, by pointing on the text and pressing the controller *trigger* button. These findings coincide with the discoveries of J. Nielsen, who lists "freedom and control" as an important heuristic for usability, see [80].
Another notice was an impatience of longer text passages. To signalize to the user how long the explanation takes, a progress bar below the message box was added. The progress bar reflects the progress of the active selected Rendering Pipeline stage, which belongs to the heuristic "Visibility of system status" in [80].

usability has been iteratively improved, but still has issues

Triangle Builder usability The usability of the *Triangle Builder* interaction was confusing (component is described in section 5.3.2). The first participants were confused how to place vertices in space. They didn't know that they has to use the "Triangle modes menu" in figure 5.4 and didn't notice the color gradient on the controllers' trackpad. Thus, a blink animation for the next awaited button and a preview (in form of a video) for each mode was added above the menu. By switching to a mode, the preview changes to show the interaction of the active interaction mode. This change affects two further heuristics, on the one hand the "visibility of the system status", but also "Recognition rather than recall –> Make the repertoire of available actions salient" [80].
After this change, the next participants were still confused how to create a triangle. But after some tries, they finally found it out and told "the interaction is only confusing, until you understand it". To create a triangle, the user has to select three vertices by hovering on them with the controller and click (press and release) the trigger button. Two of the participants tried to keep holding the trigger and "draw" a triangle with the controller between the vertices. Another confusing interaction was the selection of vertices. At this point, a deeper investigation is necessary to design the most intuitive interaction with the system. Currently, a vertex can be hovered by the colliding the controller mesh the vertex mesh. In some cases the target vertex was inside the controller and lost its' highlight because it was not colliding with the controller mesh was anymore. This effect was very noticeable and confused the participants. The color gradient, which is placed on the trackpad, was part of the iterative improvements. But also after the changes it was not recognized directly by all participants. Since this functionality has no direct effect on the Guided Tour, this point has been postponed.

interaction needs on some points assistance

Inaccurate interaction Using pointing pattern as interaction technique (section 2.2.3) was used very intuitively, but in some situations the interaction was not accurate enough. For example: the Guided Tour expected specific frustum value adjustments via moving the far and near plane and using the sliders in figure 5.5(b). Each participant completed this task, but everyone struggled to fit the values on first try. Possible improvements and ideas have been incorporated into future work.

[2] Zoom: `https://zoom.us/` (visited on 11/19/2020)

Difficulties to understand the content The learning material was implemented to provide pupils an idea of the Rendering Pipeline. The participant with pre-knowledge in CG did understand each step and only criticized the current usage of the 2D image: "it may mislead the sense of the aspect ratio". The other participants without pre-knowledge in CG understood the idea of the Rendering Pipeline, but did not understand some details, e.g. how the TRS matrix is exactly defining the world position, why do the matrices need the fourth dimension or why the inside of a 3D model is not rendered (back-face culling). Didactic investigations following this work will research on learner-centric didactical reduction.

content is still too complicated

Continuous guidance and animation All simulations and animations happened on the left or right next to the tour host. After each stage, the animations disappeared to create space for new animations. Each participant looked suddenly around the environment to check, whether they had missed something. The stages *Rasterization* and *Visibility Tests* seemed to be the most unclear stages, because there were no animations nor interactions.
Finally, each participant was confused in the end of the tour, because they awaited a "Quit" button to stop the simulation and signalize that they were ready to leave the immersive environment.

continous guidance is important

Chapter 7 Future Work and Conclusion

This chapter starts with a summary and a founded conclusion of the results of this thesis and finally ends up with some hints, needs and ideas for future work.

7.1 Future Work

To reach all visions of this Educational VR Application there is still work to be done. This section describes hints and ideas for future research and implementations.

7.1.1 Rendering Pipeline Prototype

didactic reduction of content by an expert

According to the pilot study results, the delivered learning content of the Guided Tour can help to understand, but there are still unclear parts. Thus, a didactic reduction by an expert, with school-practical experience, could be sensible to make the learning content more understandable to pupil. Based on the experts suggestions, a consideration could be to create a better adjusted version and conduct study with pupils as participants. For this study specific questions has to be made the participants have to answer after the VR experience. The questions can be based on the learning goals formalized in section 3.2.4. Further paragraphs present more detailed possible improvements.

always show something, speaking voice, term dictionary

Educational improvements At current state terms are explained as text. The results of the pilot study indicate that more integration of terms into interactions or visualization make sense. For example, it can be helpful to have kind of dictionary (menu), the user can open any time, to select terms and get a description with helpful small visualizations. As some participants were confused, when visualizations disappear completely after finishing a stage, it may be sensible to always keep a visualization active. After a didactic reduction, the explanations has to be adjusted. After doing this it may be helpful to record voices for Kyle. A speaking voice would increase the immersion, help to understand (support audio channel) and make it unnecessary to read. Thus, the learner can concentrate more on visualizations and interactions.

frustum performance, 2D preview with resized aspect ratio

Technical improvements Usability issues have to be discovered and resolved continuously. Even the frustum transformation performance issue (section 5.3.3) was not recognized in pilot study, more weak computers could have problem with it. A re-implementation of the transformation with a fragment shader may improve the performance, but then it is needed to find a solution to still track interaction with the transformed meshes (e.g. mesh approximation with less polygons). As the migration into the fragment shader allows to discard fragments inside the clip space, before normalizing them into eye space, primitives behind the camera can be respected, too. This way the issue of current clipping method, mentioned in section 6.1, will be resolved automatically. To additionally avoid confusion about the purpose of aspect ratio,

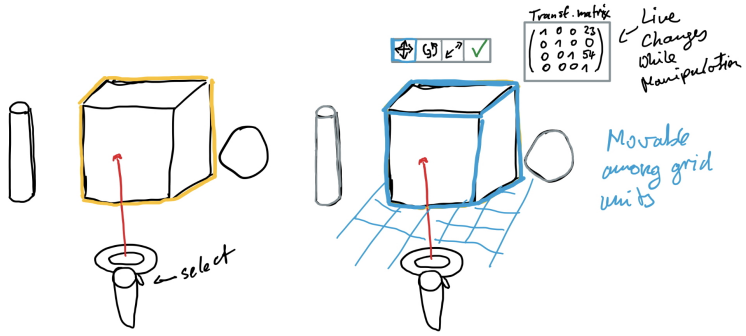
beside of the current 2D image preview (with ratio 1:1 and distorted objects inside) in front of the unit cube a second 2D image has to be added with correct aspect ratio.

"draw" triangles, spherical volume-based selection

Change Triangle Builder interaction The pilot study has shown, that the usability of building triangles is well after understanding it, but is bad in the beginning. Possible improvements are to allow pressing the trigger button while "drawing" the lines between vertices. Additional optimization would be to improve the collision between the controller and vertices. At current state, the controller mesh has to collide with the vertices exactly. This leads to confusion during usage and has to be changed. One option is to place an invisible spherical area above the controller, where the vertices has to be contained to be "selected". Another option is to use the laser pointer to select the vertices.

more visualizations for left stages, advanced shading methods

More visualizations and interactions The visualizations of the other stages (*Rasterization, Lighting, Texturing* and *Visibility Test*) are still kept simple. Supposed the hypotheses are true, more visualizations in VR would improve the user experience and enhance the learning effect. The rasterization stage could visualize the scan conversion algorithms like *Digital Diffential Analysis (DDA), Bresenham (Midpoint)* or *Pineda*. The *Lighting* stage can be extended with more, by user interactable, light and shadow simulations like *visualization of light or shadow rays, light reflection, light refraction, shadow maps, shadow volumes, global illumination* and more. The *Texturing* stage can be improved by enabling an interaction with the UV map: by pointing on the texture the selected body part (e.g. eyes) will be textured on Kyle's naked model. Finally, the *Visibility Test* could be visualized by assigning a simple colored voxel grid representing a small Z-Buffer of a small area of the scene. The voxel colors are defined by current frustum shape and selected small area. But also existing visualizations can be extended with more interactions. For example, the visualization of TRS matrix composition (figure 5.11(c)) can be made more control-able. A participant wished to point on the matrix to show what the matrix is doing (e.g. pointing on translation would move the object). Another possibility is to enable transformation options (move, rotate, scale) for game objects in general preview changes in the matrices (figure 7.1).



(a) Step 1: Select a cube ***(b)*** Step 2: Apply transformations

Figure 7.1: Transformation matrix preview

Add difficulty levels The implemented learning scenario targets the knowledge level of a *pupil* (table 3.1): no complex mathematics and visualizations to get an idea of the progress. To make it more interesting for Computer Science *students* (table 3.2), advanced shading techniques has

regulate difficulty via topic selection

to be added, without increasing the overall complexity for pupils. One possibility to achieve this is to create a more complex Rendering Pipeline scenario build for students separately. Another approach could be to regulate the difficulty level by adding sub menus for each stage. The main stage can be still simple and more complex visualization can be called by choosing them via a sub-menu.

game elements in later versions could increase the learning effectiveness

Game-based version Merchant et al. wrote in their meta-analysis [28]: "game-based learning environments were more effective than virtual worlds or simulations, with rougly twice as large". Thus, adding game-based manipulation tasks or elements inside the VR scene (e.g. achievements for completing tasks, time limits for task completion, ...) could increase the effectiveness of the learning scenario. Of course, a game-based version has to be evaluated then and compared to the experimental and guided version to check its effectiveness.

7.1.2 Educational VR Stack

This work has delivered an Educational VR Application and separated potential useful Unity assets into an own Unity package. Based on this package more Educational VR Applications has to be developed and evaluated. This way more requirements can be identified for a "Educational VR Application" and thus a "Educational VR Library".

back button for dialog

Option to "go back" in the Guided Tour Some participants had wished a "back button" to go back to the last message from Kyle. Actually, each Unity Timeline director resets all objects of the stage. To realize a "back button" each Timeline signal has to do kind of "soft reset" of actual visualization.

VRTK for transformation options

Integration of VRTK One interesting extension could be the integration of the VRTK framework (already mentioned in section 5.3.1). This is a useful toolkit for VR interactions and shall be evaluated for educational purposes followed by a migration of the delivered "EducationalVR" library. VRTK can be used, for example, to let the learner move, rotate or scale game objects. This way the learner have more possibilities to do experiments.

streaming to mobile VR in classrooms

Mobile VR Possible use case is to present the Rendering Pipeline in lectures or classrooms. In these use cases *mobile VR* solutions could be helpful to see what the lecturer can see: the lecturer (or teacher) is using this Educational VR Application as it is and the rendered image will be streamed onto mobile devices of the audience. The lecture can be hold and at the same time visualized for all directly.

seperate template manager from web socket server

VR Data Collector Server Currently, the server offers to import/export templates or to manage them on the server's local storage. Future iterations of the design-based research cycle will find out, what the persona *researcher* likes more on his panel. In case of using the server storage solution, it will be necessary to integrate a relational or NoSQL database instance. Additionally, it may make sense to split the template API (and database) from the web socket server and create a central service instance to regulate all requests between web socket server, template API and Unity client.

xAPI Definitions and Fetcher To serialize xAPI vocabulary from definitions into C# classes, the developer needs the executable of the implemented fetching tool. With .NET Core technologies the *xAPI Fetcher* can be moved to a server instance. A better alternative could be to extend this fetcher into a micro-service to translate the xAPI definitions from repository into a target programming language (JavaScript, C#, Go, C++, etc.).

fetcher as micro-service for other programming languages

7.2 Conclusion

This work had neither proved nor disproved the hypotheses *(1) "Knowledge which is achieved via experiments or visualization holds longer in long-term memory."*and *(2) "Interaction is essential for learning via VR."*, but created a first scientifically founded and evaluated prototype, which can be used for further iterations of a design-based research cycle. Further long-term and bigger scaled studies can be conducted to adjust the provided well-founded concept, which can help to prove or disprove these hypotheses. The pilot study has shown, that the prototype can help to understand several parts of the Rendering Pipeline, but is still too "difficult". This work started with the intention, that a visualization of the Rendering Pipeline in VR, may be less interesting for Computer Graphics students, but more interesting for younger students. Further discussions induced to the concrete questions *"Is it more effective to learn the Rendering Pipeline in VR as with native techniques?"* and *"Is learning via VR generally more effective?"*. The conducted evaluation has shown that these supposed questions have to be still verified with further studies. But it is sure, that the current pilot prototype can be useful for students (with CG pre-knowledge) to experience the Rendering Pipeline interactively.

can help to understand, still too "difficult"

useful for students

The implemented pilot prototype was a so called "Educational VR Application". Those kind of applications are designed to deliver VR experience for educational purposes. In combination with Learning Analytics, effective or ineffective tasks can be identified and to be improved afterwards. Most of the compared systems (section 2.3) are designed as *expositional* and *experimental* VR world applications. Thus, this VR application tried to combine both VR world classes within one Guided Tour through the Rendering Pipeline stages (section 2.1.3).

As the Rendering Pipeline has several stages, the stages can be considered as own extendable small learning scenarios. A first prototype can start with simple tasks and be extended with more content. Followed by our first intention we decided to start creating an Educational VR Application for K12 students and scale the complexity up or down if necessary.

To design a "Educational VR Application" a development progress of two cycle-based progresses was applied: design-based research and iterative design. The design-based research cycle is used to discover new didactic requirements from study results and integrate them into newer iterations. The iterative design (section 2.2) cycle was used for smaller feedback sessions. It concentrated primary on usability and technical requirements. Both follow a similar scheme and cannot be divided clearly. Based on some rules of iterative design, this thesis delivers a first complete concept of an Educational VR Application (overall vision, objective, first user stories and storyboards) for five evaluated personas (*pupil*, *student*, *teacher*, *researcher* and *developer*. The personas resulted from questionnaires, which were conducted during this thesis (chapter E). Depended on the personas (section 3.2.2) and inspired by the visions (section 3.1.1) first user stories (section 3.2.3) and requirements (section 3.3) were discussed with the supervisors. At current state, the implemented Guided Tour guides the learner through all stages. Each stage has explanations or visualizations. The learner can interact with three of the visualizations to understand some RP basics. The user can place vertices in 3D space and connect them into triangles to get an intuition for 3D geometry in general and to see how 3D meshes are built.

With manipulating the frustum, the user can understand the sense of the frustum and see the clipping mechanism.
To analyze the learning scenarios it is necessary to track users' activity. To do this the *LA Data Collector for VR*, made by A. Brocker [67], was used. During development this library was optimized and extended (section 5.4) to the new needs. This included the extension of the Unity client, web application and web socket server. The *LA Data Collector for VR* has defined xAPI statements as strings inside of the C# code. To use instead a more strictly defined xAPI vocabulary a tool was created to fetch them from a local or remote storage and translate them into C# classes (section 5.5). In agreement with the supervisors and a member of the Learning Technologies Research Group their xAPI vocabulary was used extended (section 5.5.1) by needed xAPI verbs (table B.6), activities (table B.5) and extensions (table B.7). Finally, suitable xAPI statements were applied on the implemented Guided Tour scenario (table 5.1).
During the implementation the Unity scripts were strictly seperated into generic and non-generic packages. The generic package is called "EducationalVR" and is potentially useful for new Educational VR Applications. The non-generic Unity scripts are the "main part" of the project and represent the VR application. The overall solution for the first pilot prototype of an Educational VR Application and its' connected systems is illustrated in figure 5.1.
Finally, the technical correctness of the implemented solution was evaluated in section 6.1 and a pilot study (section 6.2)) was conducted to discover first issues and reactions. The study have found some bugs and usability issues, which were resolved immediately after each participant test sessions. Four of the participants were experts and they provided hints for further extensions.

Appendix

Appendix A Bibliography

[1] T. Akenine-Möller, E. Haines, and N. Hoffman, *Real-Time Rendering*, en, Third. A K Peters/CRC Press, Apr. 2012, ISBN: 978-1-56881-424-7. DOI: 10.1201/9781315365459.

[2] C. T. Murniati and R. Sanjaya, *Learning Technologies in Education: Issues and Trends*, en. Semarang - Indonesia: Soegijapranata Catholic University, 2017, ISBN: 978-602-686-540-3.

[3] V. Arkorful and N. Abaidoo, "The role of e-learning, the advantages and disadvantages of its adoption in Higher Education.," en, vol. 2, no. 12, p. 14, 2014.

[4] J. O'Brien, "2020 EDUCAUSE Horizon Report | Teaching and Learning Edition," en, p. 58, 2020.

[5] J. Martín-Gutiérrez, C. E. Mora, B. Añorbe-Díaz, and A. González-Marrero, "Virtual Technologies Trends in Education," en, *EURASIA Journal of Mathematics, Science & Technology Education*, vol. 13, no. 2, pp. 469–486, Feb. 2017, ISSN: 1305-8223. DOI: 10.12973/eurasia.2017.00626a.

[6] I. Stojšić, A. Ivkov-Džigurski, and O. Maričić, "Virtual Reality as a Learning Tool: How and Where to Start with Immersive Teaching," en, in *Didactics of Smart Pedagogy*, L. Daniela, Ed., Cham: Springer International Publishing, 2019, pp. 353–369, ISBN: 978-3-030-01550-3 978-3-030-01551-0. DOI: 10.1007/978-3-030-01551-0_18.

[7] V. S. Pantelidis, "Reasons to Use Virtual Reality in Education and Training Courses and a Model to Determine When to Use Virtual Reality," en, p. 12, Jan. 2009.

[8] H.-M. Huang, U. Rauch, and S.-S. Liaw, "Investigating learners' attitudes toward virtual reality learning environments: Based on a constructivist approach," en, *Computers & Education*, vol. 55, no. 3, pp. 1171–1182, Nov. 2010, ISSN: 03601315. DOI: 10.1016/j.compedu.2010.05.014.

[9] N. Mustafa, "IMPACT OF THE 2019–20 CORONAVIRUS PANDEMIC ON EDUCATION," en, p. 12, 2020.

[10] P. Gouëdard, B. Pont, and R. Viennet, "Education responses to COVID-19: Implementing a way forward," en, OECD Education Working Papers 224, Jul. 2020. DOI: 10.1787/8e95f977-en.

[11] M. Dunleavy, C. Dede, and R. Mitchell, "Affordances and Limitations of Immersive Participatory Augmented Reality Simulations for Teaching and Learning," en, *Journal of Science Education and Technology*, vol. 18, no. 1, pp. 7–22, Feb. 2009, ISSN: 1059-0145, 1573-1839. DOI: 10.1007/s10956-008-9119-1.

[12] D. Kamińska, T. Sapiński, S. Wiak, *et al.*, "Virtual Reality and Its Applications in Education: Survey," en, *Information*, vol. 10, no. 10, p. 20, Oct. 2019, ISSN: 2078-2489. DOI: 10.3390/info10100318.

[13] N. Law, D. S. Niederhauser, R. Christensen, and L. Shear, "A Multilevel System of Quality Technology-Enhanced Learning and Teaching Indicators," en, *Educational Technology & Society*, vol. 19, no. 3, pp. 72–83, 2016, ISSN: 1436-4522.

[14] A. Kirkwood and L. Price, "Technology-enhanced learning and teaching in higher education: What is 'enhanced' and how do we know? A critical literature review," en, *Learning, Media and Technology*, vol. 39, no. 1, pp. 6–36, Jan. 2014, ISSN: 1743-9884, 1743-9892. DOI: 10.1080/17439884.2013.770404.

[15] A. Algahtani, "Evaluating the Effectiveness of the E-learning Experience in Some Universities in Saudi Arabia from Male Students Perceptions," en, Doctoral Thesis, Durham University, 2011.

[16] M. Muhajirah, "Basic of Learning Theory: (Behaviorism, Cognitivism, Constructivism, and Humanism)," en, *International Journal of Asian Education*, vol. 1, no. 1, pp. 37–42, Jun. 2020, ISSN: 2722-8592. DOI: 10.46966/ijae.v1i1.23.

[17] R. Huang, J. M. Spector, and J. Yang, *Educational Technology: A Primer for the 21st Century*, en, ser. Lecture Notes in Educational Technology. Singapore: Springer Singapore, 2019, ISBN: 9789811366420 9789811366437. DOI: 10.1007/978-981-13-6643-7.

[18] A. Glassner and S. Back, *Exploring Heutagogy in Higher Education: Academia Meets the Zeitgeist*, en. Singapore: Springer Singapore, 2020, ISBN: 9789811541438 9789811541445. DOI: 10.1007/978-981-15-4144-5.

[19] K. Carlson, "Brief Paper on Connectivism Literature," en, 2020. DOI: 10.13140/RG.2.2.25033.98402.

[20] M. D. Merrill, "First principles of instruction," en, *Educational Technology Research and Development*, vol. 50, no. 3, pp. 43–59, Sep. 2002, ISSN: 1042-1629, 1556-6501. DOI: 10.1007/BF02505024.

[21] M. D. Kickmeier-Rust and D. Albert, "Learning Analytics to Support the Use of Virtual Worlds in the Classroom," en, in *Human-Computer Interaction and Knowledge Discovery in Complex, Unstructured, Big Data*, A. Holzinger and G. Pasi, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2013, pp. 358–365, ISBN: 978-3-642-39146-0. DOI: 10.1007/978-3-642-39146-0_34.

[22] M. A. Chatti, A. L. Dyckhoff, U. Schroeder, and H. Thüs, "A reference model for learning analytics," en, *International Journal of Technology Enhanced Learning*, vol. 4, no. 5/6, p. 318, 2012, ISSN: 1753-5255, 1753-5263. DOI: 10.1504/IJTEL.2012.051815.

[23] V. Camilleri, S. de Freitas, M. Montebello, and P. McDonagh-Smith, "A case study inside virtual worlds: Use of analytics for immersive spaces," en, in *Proceedings of the Third International Conference on Learning Analytics and Knowledge - LAK '13*, Leuven, Belgium: ACM Press, 2013, p. 230, ISBN: 978-1-4503-1785-6. DOI: 10.1145/2460296.2460341.

[24] M. Ehlenz, B. Heinemann, T. Leonhardt, R. Röpke, V. Lukarov, and U. Schroeder, "Eine forschungspraktische Perspektive auf xAPI-Registries," de, in *DELFI 2020 – Die 18. Fachtagung Bildungstechnologien der Gesellschaft für Informatik e.V.*, Bonn: Gesellschaft für Informatik e.V., 2020, p. 6, ISBN: 978-3-88579-702-9.

[25] V. Lukarov, M. Ehlenz, and U. Schroeder, "Building a data warehouse for multimodal learning analytics research projects," in *Companion Proceedings of the Tenth International Conference on Learning Analytics & Knowledge*, Society for Learning Analytics Research, 2020, pp. 25–28.

[26] A. Hussien, "A Review on E-Learning Technology," en, vol. 2, no. 1, p. 3, Dec. 2019.

[27] X. Lei, A. Zhang, B. Wang, and P.-L. P. Rau, "Can Virtual Reality Help Children Learn Mathematics Better? The Application of VR Headset in Children's Discipline Education," en, in *Cross-Cultural Design. Applications in Cultural Heritage, Creativity and Social Development*, P.-L. P. Rau, Ed., vol. 10912, Cham: Springer International Publishing, 2018, pp. 60–69, ISBN: 978-3-319-92251-5 978-3-319-92252-2. DOI: 10.1007/978-3-319-92252-2_5.

[28] Z. Merchant, E. T. Goetz, L. Cifuentes, W. Keeney-Kennicutt, and T. J. Davis, "Effectiveness of virtual reality-based instruction on students' learning outcomes in K-12 and higher education: A meta-analysis," en, *Computers & Education*, vol. 70, pp. 29–40, Jan. 2014, ISSN: 03601315. DOI: 10.1016/j.compedu.2013.07.033.

[29] X. Dong, "An overall solution of Virtual Reality classroom," en, in *2016 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*, Beijing, China: IEEE, Jul. 2016, pp. 119–123, ISBN: 978-1-5090-2927-3. DOI: 10.1109/SOLI.2016.7551672.

[30] H. Kaufmann, "Collaborative Augmented Reality in Education," en, in *Imagina Conference 2003*, Monaco, Mar. 2003.

[31] G. Burdea and P. Coiffet, *Virtual Reality Technology*, eng, 2nd ed. Hoboken, N.J: J. Wiley-Interscience, 2003, ISBN: 978-0-471-36089-6.

[32] R. Zender and M. Weise, "Lehren und Lernen mit VR und AR – Was wird erwartet? Was funktioniert?" de, Sep. 2018, p. 13.

[33] D. Vergara, J. Extremera, M. P. Rubio, and L. P. Dávila, "Meaningful Learning Through Virtual Reality Learning Environments: A Case Study in Materials Engineering," en, *Applied Sciences*, vol. 9, no. 21, p. 4625, Oct. 2019, ISSN: 2076-3417. DOI: 10.3390/app9214625.

[34] L. Freina, "Immersive vs Desktop Virtual Reality in Game Based Learning," en, in *9th European Conference on Games Based Learning (ECGBL)*, Oct. 2015.

[35] E. R. Black, "Learning Then and There: An Exploration of Virtual Reality in K-12 History Education," en, Ph.D. dissertation, Dec. 2017.

[36] M. Weise and R. Zender, "Interaktionstechniken in VR-Lernwelten," in *DeLFI Workshops 2017*, Chemnitz, Sep. 2017.

[37] S. Schwan and J. Buder, "Lernen und Wissenserwerb in virtuellen Realitäten," ger, in *Virtuelle Realitäten*, ser. Internet und Psychologie 5, Göttingen: Hogrefe, Verl. für Psychologie, 2002, pp. 109–132, ISBN: 978-3-8017-1465-9.

[38] M. Prins, "TogetherVR: A framework for photo-realistic shared media experiences in 360-degree VR," en, in *IBC Conference 2017*, Amsterdam, Sep. 2017, p. 9.

[39] B. Meunier and J. Cosmas, "5G Internet of Radio Light Virtual Reality System," en, in *2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, Valencia: IEEE, Jun. 2018, pp. 1–5, ISBN: 978-1-5386-4729-5. DOI: 10.1109/BMSB.2018.8436602.

[40] L. Shi, D. Shi, X. Zhang, *et al.*, "5G Internet of Radio Light positioning system for indoor broadcasting service," en, p. 12, Apr. 2020.

[41] L. Jensen and F. Konradsen, "A review of the use of virtual reality head-mounted displays in education and training," en, *Education and Information Technologies*, vol. 23, no. 4, pp. 1515–1529, Jul. 2018, ISSN: 1360-2357, 1573-7608. DOI: 10.1007/s10639-017-9676-0.

[42] A. Wiepke, E. Richter, R. Zender, and D. Richter, "Einsatz von Virtual Reality zum Aufbau von Klassenmanagement-Kompetenzen im Lehramtsstudium," de, 2019, ISSN: 1617-5468. DOI: 10.18420/DELFI2019_319.

[43] J. Jerald, *The VR Book: Human-Centered Design for Virtual Reality*. Association for Computing Machinery and Morgan & Claypool, 2015, ISBN: 978-1-970001-12-9.

[44] E. Chang, H. T. Kim, and B. Yoo, "Virtual Reality Sickness: A Review of Causes and Measurements," en, *International Journal of Human–Computer Interaction*, vol. 36, no. 17, pp. 1658–1682, Oct. 2020, ISSN: 1044-7318, 1532-7590. DOI: 10.1080/10447318.2020.1778351.

[45] J. J. LaViola, "A discussion of cybersickness in virtual environments," en, *ACM SIGCHI Bulletin*, vol. 32, no. 1, pp. 47–56, Jan. 2000, ISSN: 0736-6906. DOI: 10.1145/333329.333344.

[46] L. Velho, J. Giannella, L. Carvalho, and D. Lucio, "VR TOUR: GUIDED PARTICIPATORY META-NARRATIVE FOR VIRTUAL REALITY EXPLORATION," en, p. 9, Sep. 2019.

[47] P. Shirley and S. Marschner, *Fundamentals of Computer Graphics*, 3rd ed. Natick, Mass: A K Peters, 2009, ISBN: 978-1-56881-469-8.

[48] L. Kobbelt, *Basic Techniques in Computer Graphics: Rendering Pipeline (Lecture Notes)*, en, RWTH Aachen, Computer Graphics Lecture, Aachen, 2019.

[49] E. Angel and D. Shreiner, "An introduction to shader-based OpenGL programming," en, in *ACM SIGGRAPH 2009 Courses on - SIGGRAPH '09*, New Orleans, Louisiana: ACM Press, 2009, pp. 1–152. DOI: 10.1145/1667239.1667245.

[50] T. C. Reeves and A. Hall, "Enhancing the Worth of Instructional Technology Research through "Design Experiments" and Other Development Research Strategies," en, p. 15, Apr. 2000.

[51] T. Anderson and J. Shattuck, "Design-Based Research: A Decade of Progress in Education Research?" en, *Educational Researcher*, vol. 41, no. 1, pp. 16–25, Jan. 2012, ISSN: 0013-189X, 1935-102X. DOI: 10.3102/0013189X11428813.

[52] J. K. Shah, D. C. Ensminger, and K. Thier, "The Time for Design-Based Research is Right and Right Now," en, vol. 27, no. 2, p. 22, Feb. 2020.

[53] G. E. Lefoe, M. O'Reilly, D. R. Parrish, S Bennett, M Keppell, and C Gunn, "The Carrick Exchange: Not Just Another Repository," en, p. 8, Jun. 2007.

[54] R. Mcmahan and D. Bowman, "An Empirical Comparison of Task Sequences for Immersive Virtual Environments," in *2007 IEEE Symposium on 3D User Interfaces*, Charlotte, NC, USA: IEEE, 2007, p. 4 142 841, ISBN: 978-1-4244-0907-5. DOI: 10.1109/3DUI.2007.340770.

[55] S. Razzaque, Z. Kohn, and M. C. Whitton, "Redirected Walking," in *EUROGRAPHICS 2001*, North Carolina, Aug. 2001.

[56] Y. Yair, R. Mintz, and S. Litvak, "3D-Virtual Reality in Science Education: An Implication for Astronomy Teaching," en, vol. 20, no. 3, pp. 293–305, 2001.

[57] Y. Yair, Y. Schur, and R. Mintz, "A "Thinking Journey" to the Planets Using Scientific Visualization Technologies: Implications to Astronomy Education," en, *Journal of Science Education and Technology*, vol. 12, no. 1, pp. 43–49, 2003, ISSN: 10590145. DOI: 10.1023/A:1022107627775.

[58] H. Kaufmann and D. Schmalstieg, "Designing Immersive Virtual Reality for Geometry Education," in *IEEE Virtual Reality Conference (VR 2006)*, Mar. 2006, pp. 51–58. DOI: 10.1109/VR.2006.48.

[59] H. Kaufmann and D. Schmalstieg, "Mathematics and geometry education with collaborative augmented reality," in *ACM SIGGRAPH 2002 Conference Abstracts and Applications*, ser. SIGGRAPH '02, San Antonio, Texas: Association for Computing Machinery, Jul. 2002, pp. 37–41, ISBN: 978-1-58113-525-1. DOI: 10.1145/1242073.1242086.

[60] H. Kaufmann, D. Schmalstieg, and M. Wagner, "Construct3D: A Virtual Reality Application for Mathematics and Geometry Education," en, *Education and Information Technologies*, vol. 5, no. 4, pp. 263–276, Dec. 2000, ISSN: 1573-7608. DOI: 10.1023/A:1012049406877.

[61] G. Taxén and A. Naeve, "CyberMath: A Shared Virtual Environment for Mathematics Exploration," Nov. 2001.

[62] C. Carlsson and O. Hagsand, "DIVE A multi-user virtual reality system," en, in *Proceedings of IEEE Virtual Reality Annual International Symposium*, Seattle, WA, USA: IEEE, 1993, pp. 394–400, ISBN: 978-0-7803-1363-7. DOI: 10.1109/VRAIS.1993.380753.

[63] A. M. Spalter, P. A. Stone, B. J. Meier, T. S. Miller, and R. M. Simpson, "Interaction in an IVR Museum of Color: Constructivism Meets Virtual Reality," en, *Leonardo*, vol. 35, no. 1, pp. 87–90, Feb. 2002, ISSN: 0024-094X, 1530-9282. DOI: 10.1162/002409402753689380.

[64] L. Stefan, F. Moldoveanu, and D. Gheorghiu, "Evaluating a Mixed-Reality 3D Virtual Campus With Big Data and Learning Analytics: A transversal study," en, *Journal of e-Learning and Knowledge Society*, vol. Vol 12, Cognitive Computing and Innovative Teaching Models, May 2016. DOI: 10.20368/1971-8829/1132.

[65] F.-P. Lin and P. Stanley, *Avataaars Generator*, en, https://getavataaars.com/, Avatar Generator, Free for Personal and Commercial Use.

[66] A. Mehrfard, J. Fotouhi, G. Taylor, T. Forster, N. Navab, and B. Fuerst, "A Comparative Analysis of Virtual Reality Head-Mounted Display Systems," en, *arXiv:1912.02913 [cs]*, Dec. 2019. arXiv: 1912.02913 [cs].

[67] A. Brocker, "A Learning Analytics Data Collector for Virtual Reality Using Unity and xAPI," en, M.S. thesis, RWTH Aachen University, Aachen, Mar. 2020.

[68] J. Linowes, *UNITY VIRTUAL REALITY PROJECTS -: Create Interactive vr Projects with Unity 2018*. en. Place of publication not identified: PACKT Publishing Limited, 2018, ISBN: 978-1-78847-880-9.

[69] P. Neg, *How are 10 Global Companies Using Node.js in Production?* en, https://www.tothenew.com/blog/how-are-10-global-companies-using-node-js-in-production/, Blog, Jun. 2017.

[70] S. Patni, "Fundamentals of RESTful APIs," en, in *Pro RESTful APIs*. Berkeley, CA: Apress, 2017, pp. 1–9, ISBN: 978-1-4842-2664-3 978-1-4842-2665-0. DOI: 10.1007/978-1-4842-2665-0_1.

[71] Á. Serrano-Laguna, I. Martínez-Ortiz, J. Haag, D. Regan, A. Johnson, and B. Fernández-Manjón, "Applying standards to systematize learning analytics in serious games," en, *Computer Standards & Interfaces*, vol. 50, pp. 116–123, Feb. 2017, ISSN: 09205489. DOI: 10.1016/j.csi.2016.09.014.

[72] A. Johnson, A. Downes, J. Haag, A. Yang, and S. Castro, *xAPI Specification*, https://github.com/adlnet/xAPI-Spec/blob/master/xAPI-About.md#def-learning-record-store, Github Repository, Apr. 2017.

[73] S. Daityari, *Angular vs React vs Vue: Which Framework to Choose in 2020*, en, https://www.codeinwp.com/blog/angular-vs-vue-vs-react/, Sep. 2020.

[74] R. Chopra, *Software Quality Assurance: A Self-Teaching Introduction*, en. Dulles, Virginia: Mercury Learning and Information, 2018, ISBN: 978-1-68392-168-4.

[75] J. Zhang, S. Li, J.-Y. Zhang, F. Du, Y. Qi, and X. Liu, "A Literature Review of the Research on the Uncanny Valley," en, in *Cross-Cultural Design. User Experience of Products, Services, and Intelligent Environments*, P.-L. P. Rau, Ed., vol. 12192, Cham: Springer International Publishing, 2020, pp. 255–268, ISBN: 978-3-030-49787-3 978-3-030-49788-0. DOI: 10.1007/978-3-030-49788-0_19.

[76] M. Mori, K. MacDorman, and N. Kageki, "The Uncanny Valley [From the Field]," en, *IEEE Robotics & Automation Magazine*, vol. 19, no. 2, pp. 98–100, Jun. 2012, ISSN: 1070-9932. DOI: 10.1109/MRA.2012.2192811.

[77] K. A. Ericsson and H. A. Simon, *Protocol Analysis: Verbal Reports as Data*, ser. Protocol Analysis: Verbal Reports as Data. Cambridge, MA, US: The MIT Press, 1984, p. 426, ISBN: 978-0-262-55012-3.

[78] T. Grossman, G. Fitzmaurice, and R. Attar, "A Survey of Software Learnability: Metrics, Methodologies and Guidelines," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '09, New York, NY, USA: ACM, 2009, pp. 649–658, ISBN: 978-1-60558-246-7. DOI: 10.1145/1518701.1518803.

[79] C. Fussan and V. Hanesova, *Pre-Testing der Lerneffektivität von 2D-to-3D-Didactics in immersiven VR-Umgebungen*, de. Gesellschaft für Informatik e.V., 2020, ISBN: 978-3-88579-702-9.

[80] J. Nielsen, "Enhancing the explanatory power of usability heuristics," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '94, New York, NY, USA: Association for Computing Machinery, Apr. 1994, pp. 152–158, ISBN: 978-0-89791-650-9. DOI: 10.1145/191666.191729.

Appendix B Tables

Pattern	When to use	Limitations
Hand Selection	realistic interactions	physiology limits, user has to first travel to the object can occlude objects
Pointing	typically better for selection, small hand motions, faster for remote, more precisely for close objects	not appropriate when realistic interaction is required, difficult to select far small objects
Image-Plane Selection	works well at any distance	works well for single eye can occlude objects
Volume-Based Selection	selection of a volume of space like voxels, pixel clouds, geometrics surfaces, etc.	rough, can be more challenging

Table B.1: Comparison of selection types. Summarized from [43].

Controller	Description
Action Controller	Listens for controller actions.
Future Trackable Controller	List of future trackable names.
Gaze Controller	Logs for eye and gaze actions[1].
Head Controller	Logs head movements[1].
Interaction Controller	Listens for virtual hands interactions[2].
Laser Pointer Controller	Listens for laser pointer interactions.
Playable Director Tracking Controller	Logs playable director events.
Teleport Controller	Listens for teleport actions.

[1] *Functionality were migrated from [67].*

[2] *Not implemented yet and is only used as placeholder.*

Table B.2: Tracking Controllers

Event	Arguments	**Description**
:init	client: String	Tells the client type (e.g. unity, panel). (Or over handshake headers entry "clientType").
:all	type: String	Get all online clients by a specific type.
:quit	ids: [String]	Close all clients by the provided ids.
:setup	ids: [String] template: Template	Setup template to all provided clients.
:meta	isCalibrated: Boolean isTracking: Boolean isCalibrating: Boolean computerName: String	Sends status information about the connected client.
:scenario	name: String gameObjects: [String] actions: [String] gestures: [String]	Provide scenario information.
:calibration:start	ids: [String]	Starts calibration of provided clients.
:calibration:stop	ids: [String]	Stops calibration of provided clients.
:tracking:start	ids: [String]	Starts tracking session of provided clients.
:tracking:stop	ids: [String]	Stops tracking session of provided clients.

Table B.3: List of socket server clients events

HTTP Method	**Endpoint**	**Description**
GET	*/api/templates/*	List all stored templates as array.
	/api/templates/:id	List template with the id ":id"
POST	*/api/templates*	Create a new template.
PUT	*/api/templates/:id*	Update template with id ":id"
DELETE	*/api/templates*	Delete template with id ":id"

Table B.4: CRUD endpoints for templates

Context	Activity	Description
eyeTracking	eye	Actors physical (left or right) eye.
gestures	hand	Actors physical (left or right) hand.
	head	Actors physical head.
virtualReality	action	An action the actor can trigger.
	avatar	An avatar in a VR environment.
	controller button	A VR controller button the actor can use.
	pointable	A VR object which can be pointed with a controller laser.
	teleport point	A fixed teleport point in VR environment.
	touchpad	A VR controller touchpad the actor can use.
	vr object	A virtual object in a VR environment.

Table B.5: New xAPI activities

Context	Verb	Description
eyeTracking	fixated	An actor fixated an object with her eyes.
	focused	An actor focused an object with her eyes.
gestures	gazed	An actor has gazed a part of her body in a direction.
	nodded	An actor has nodded part of her head.
	shaked	An actor has shaked part of her body.
	used	An actor has used part of her body.
virtualReality	created	An actor has created a virtual object in the VR environment.
	interacted	An actor interacted with the VR object in the VR environment.
	moved	An actor has changed a virtual object's position in VR environment.
	placed	An actor has placed a virtual object in the VR environment.
	pointed	An actor has pointed on an VR object.
	pressed	The actor has pressed a controller button.
	released	The actor has released a controller button.
	removed	An actor has removed a virtual object from the VR environment.
	rotated	An actor has changed a virtual object's rotation in VR environment.
	scaled	An actor has changed a virtual object's scale in VR environment.
	teleported	An actor has changed her own position in VR environment by teleport.
	touched	An actor has changed a virtual object's scale in VR environment.

Table B.6: New xAPI verbs

Context	Type	Extension	Description
eyeTracking	activity	eye	Actors physical (left or right) eye.
	result	number of blinking	The number of actors eye blinking.
generic	activity	color	The color of an object.
	result	index	It is a number to indicies something.
		time span	A time span depending on context.
gestures	activity	hand	Hand which is currently active.
	result	number of gestures	The number of actors gestures.
		primary hand	Setting, which hand is currently the primary hand.
systemControl	activity	name	Name of a system control activity. Must be a string.
virtualReality	activity	action name	It is the name of an action.
		VR object name	Name of a VR object. Must be a string.

Table B.7: New xAPI extensions

Appendix C Listings

```
{
  "actor": {
    "name": "Sally Glider",
    "mbox": "mailto:sally@example.com"
  },
  "verb": {
    "id": "http://adlnet.gov/expapi/verbs/completed",
    "display": { "en-US": "completed" }
  },
  "object": {
    "id": "http://example.com/activities/hang-gliding-test",
    "definition": {
      "type": "http://adlnet.gov/expapi/activities/assessment",
      "name": { "en-US": "Hang Gliding Test" },
      "description": {
        "en-US": "The Solo Hang Gliding test, consisting of a timed flight from
             the peak of Mount Magazine"
      },
      "extensions": {
        "http://example.com/gliderClubId": "test-435"
      }
    }
  },
  "result": {
    "completion": true,
    "success": true,
    "score": {
      "scaled": 0.95
    },
    "extensions": {
      "http://example.com/flight/averagePitch": 0.05
    }
  },
  "context": {
    "instructor": {
      "name": "Irene Instructor",
      "mbox": "mailto:irene@example.com"
    },
    "contextActivities":{
      "parent": { "id": "http://example.com/activities/hang-gliding-class-a" }
      "grouping": { "id": "http://example.com/activities/hang-gliding-school" }
    },
```

```
    "extensions": {
      "http://example.com/weatherConditions": "rainy"
    }
  },
  "timestamp": "2012-07-05T18:30:32.360Z",
  "stored": "2012-07-05T18:30:33.540Z",
  "authority": {
    "name": "Irene Instructor",
    "mbox": "mailto:irene@example.com"
  }
}
```

Listing C.1: xAPI statement complete example from https://xapi.com/statements-101/, retreived 09/30/2020

```
{
  "name": {
    "en-US": "interacted",
    "de-DE": "interagierte"
  },
  "description": {
    "en-US": "An actor interacted with the VR object in the VR environment.",
    "de-DE": "Eine Akteurin interagierte einem VR Objekt in der VR Umgebung."
  }
}
```

Listing C.2: xAPI verb interacted

```
// File server.js
...
if ('client' in jsonMessage) {
  //localeCompare: 0 = match, 1 = no match
  if (jsonMessage.client.localeCompare('Browser') === 0) {
    console.log("Browser Opened");
    lookup["Browser"] = wsServer;
  readAndSendAllInfosCSV();
  } else if (jsonMessage.client.localeCompare('Unity') === 0) {
    console.log("Unity Opened");
    lookup["Unity"] = wsServer;
  }
}
if ('setupUnity' in jsonMessage) {
  console.log("Send to Unity");
  lookup["Unity"].send(JSON.stringify(jsonMessage['setupUnity']));
}
...
```

Listing C.3: Code snippet of old socket handling

Appendix D Figures

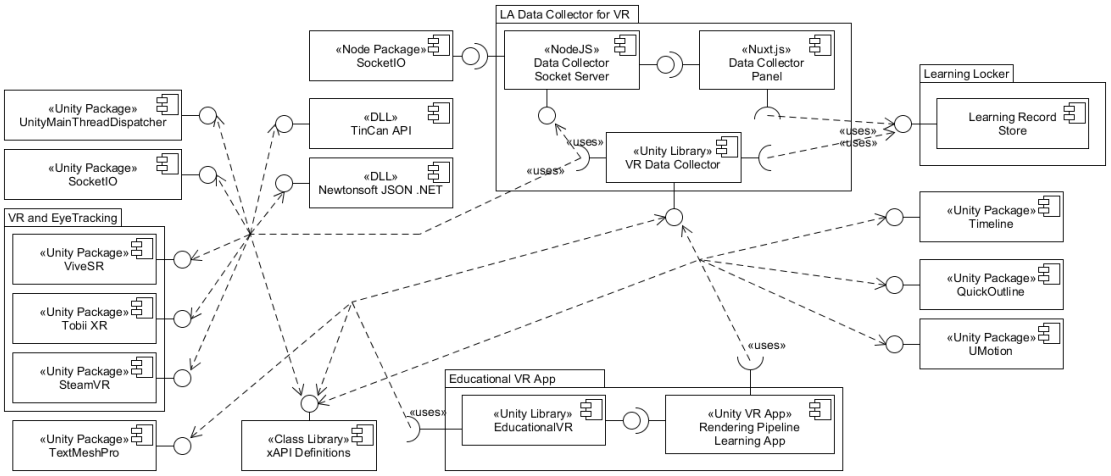


Figure D.1: Component Diagram of the system

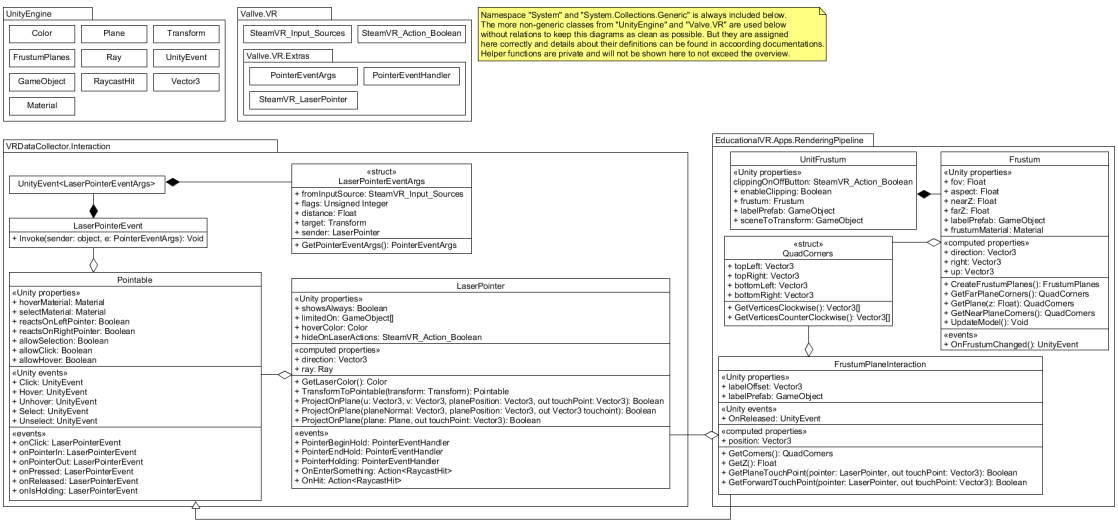
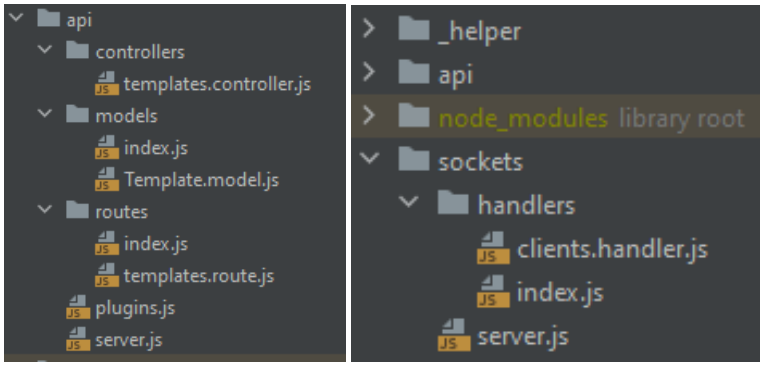
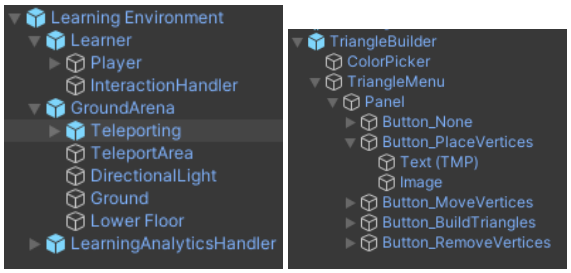


Figure D.2: Class diagram for Frustum implementation



(a) REST API structure *(b)* Socket server structure

Figure D.3: New LA Data Collector server structures



(a) Learning Environment *(b)* Triangle Builder

Figure D.4: EducationalVR Prefabs

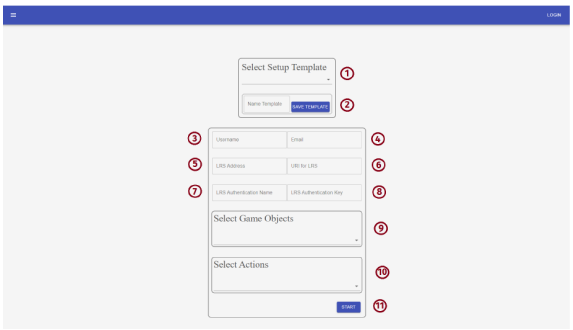


Figure D.5: Old VR LA Data Collector frontend for tracking configuration [67]

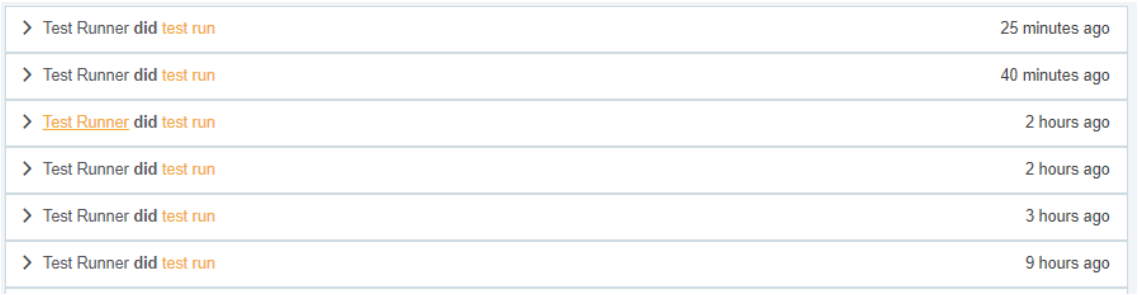
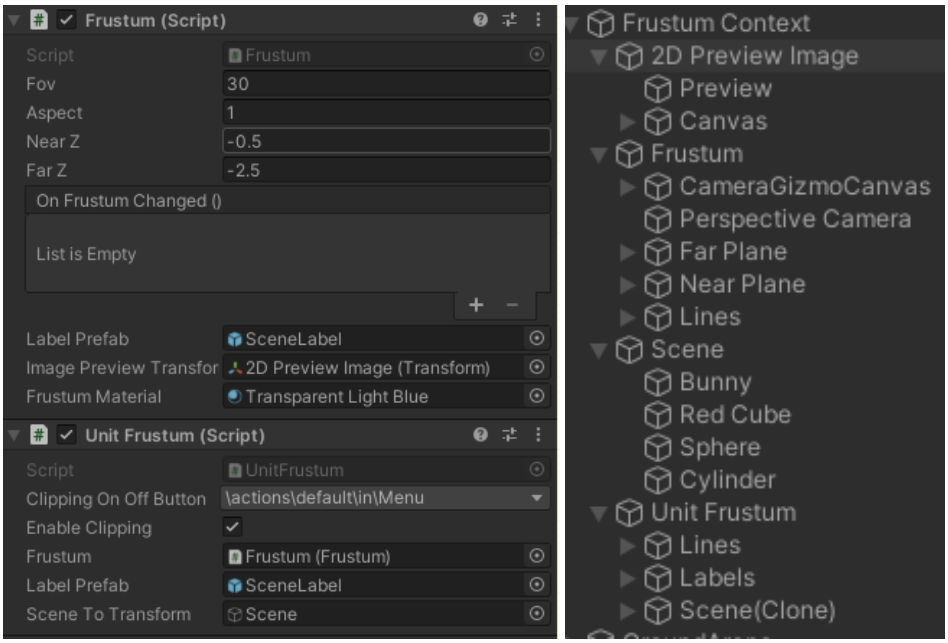


Figure D.7: Technical Evaluation of xAPI statement forwarding.



(a) Unity components of Frustum and Unit Frustum *(b)* Game object "Frustum Context"

Figure D.6: Implementation Frustum and Unit Frustum

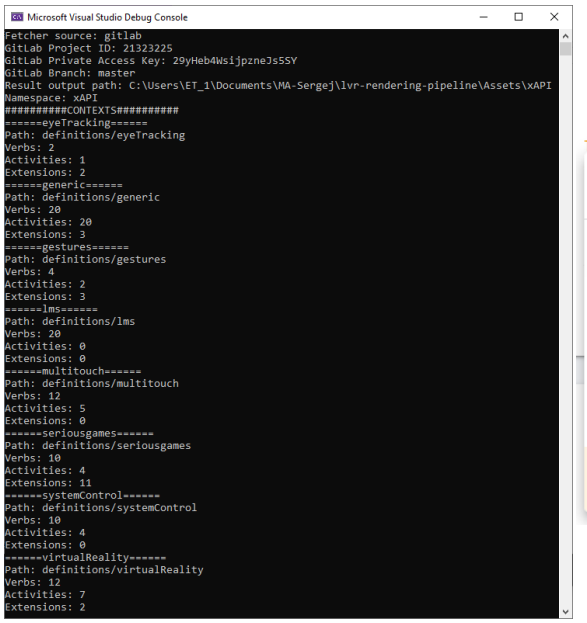


Figure D.8: xAPI Definitions Fetcher User Input[1]

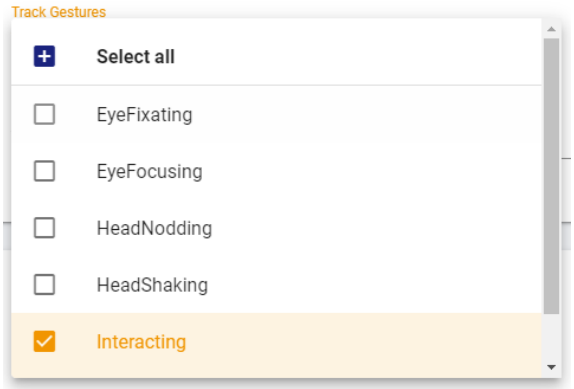


Figure D.9: Compact Select Box

[1] GitLab's private access key is used as example and is for security reasons not valid anymore. The Project ID is private also not available in public.

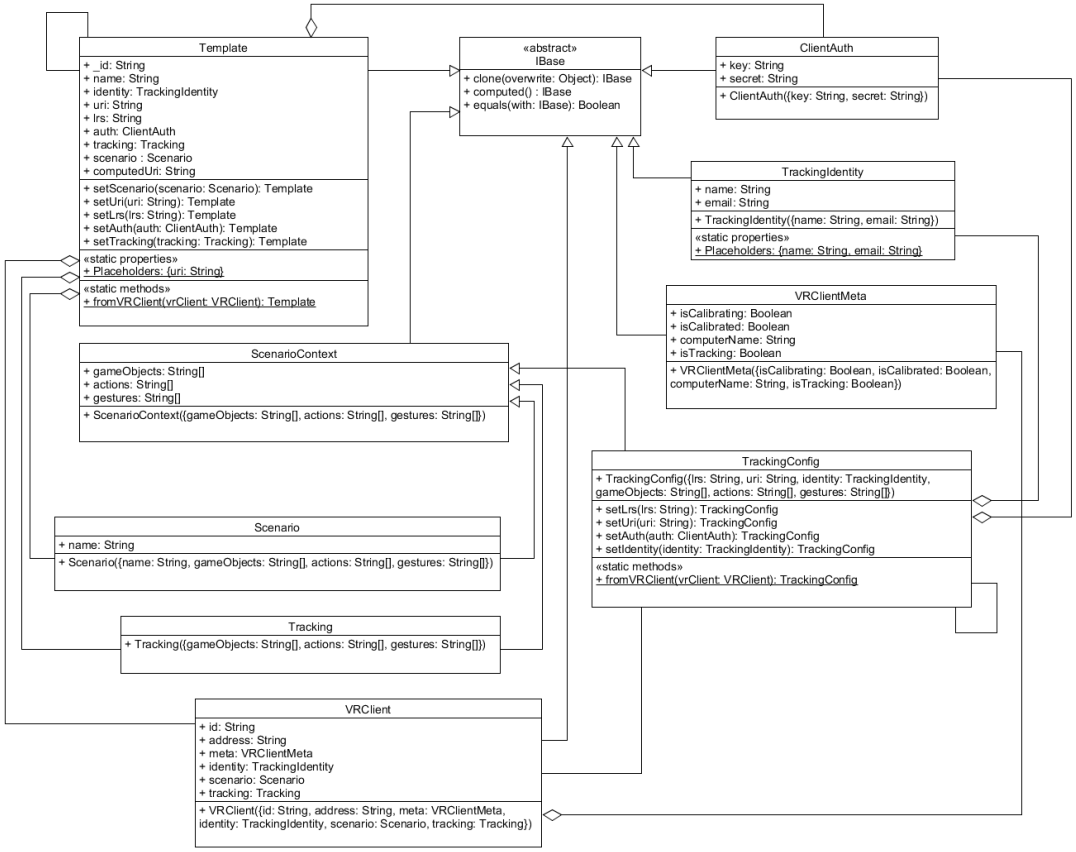


Figure D.10: Frontend data classes

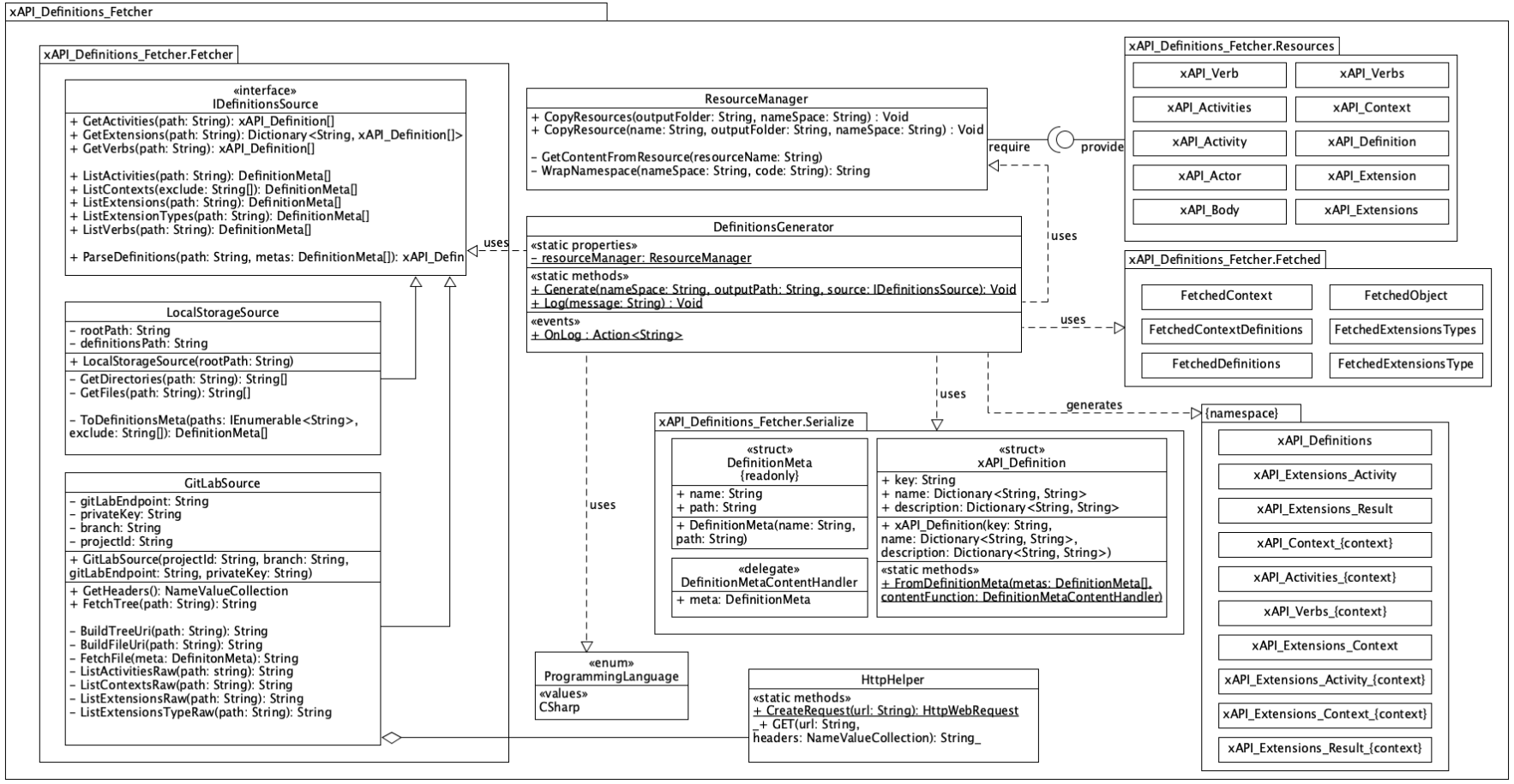


Figure D.11: xAPI Definitions Fetcher Class Diagram. More class details in figure D.12 and D.13

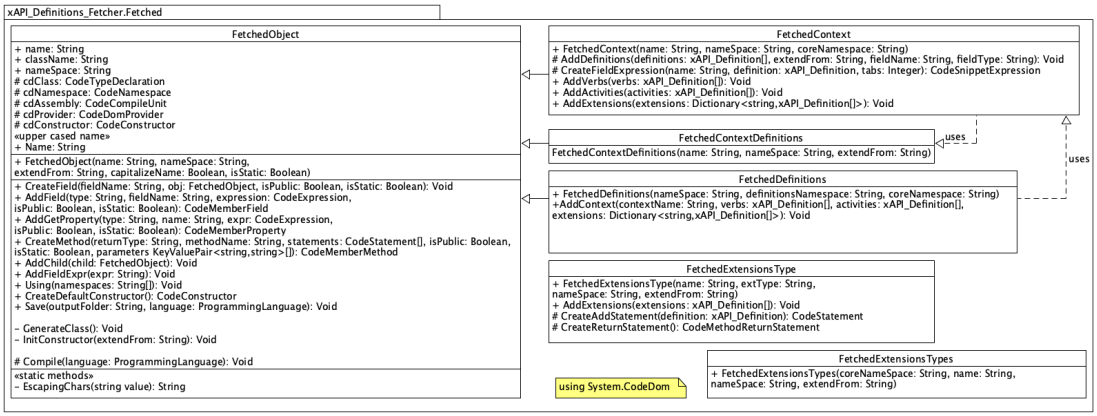


Figure D.12: xAPI Definitions Fetcher Fetched Classes

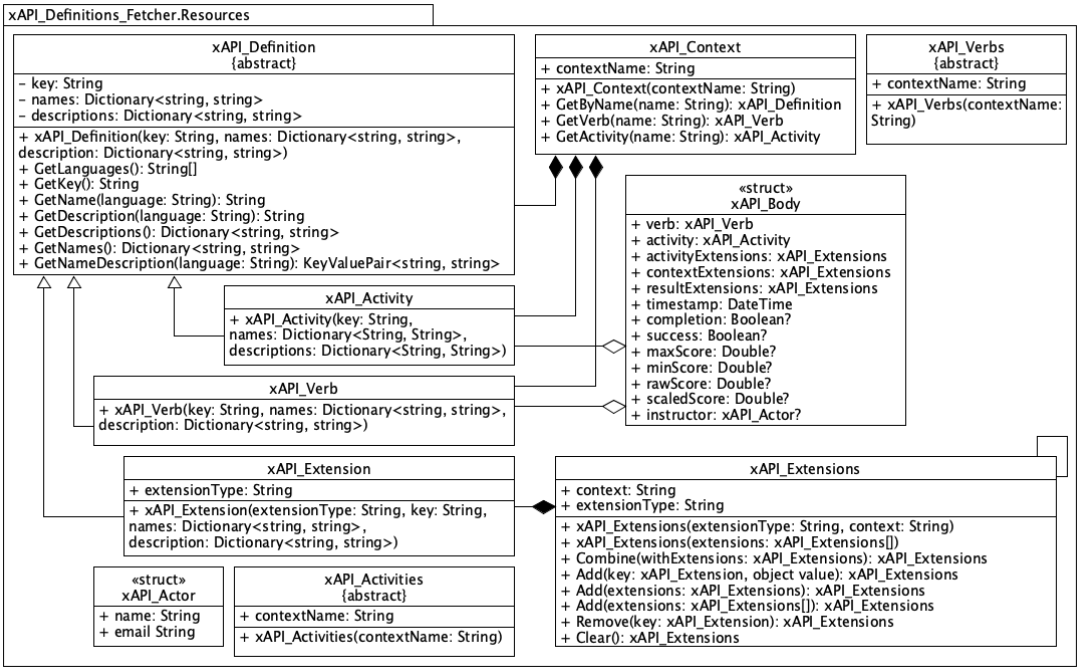


Figure D.13: xAPI Definitions Fetcher Resources

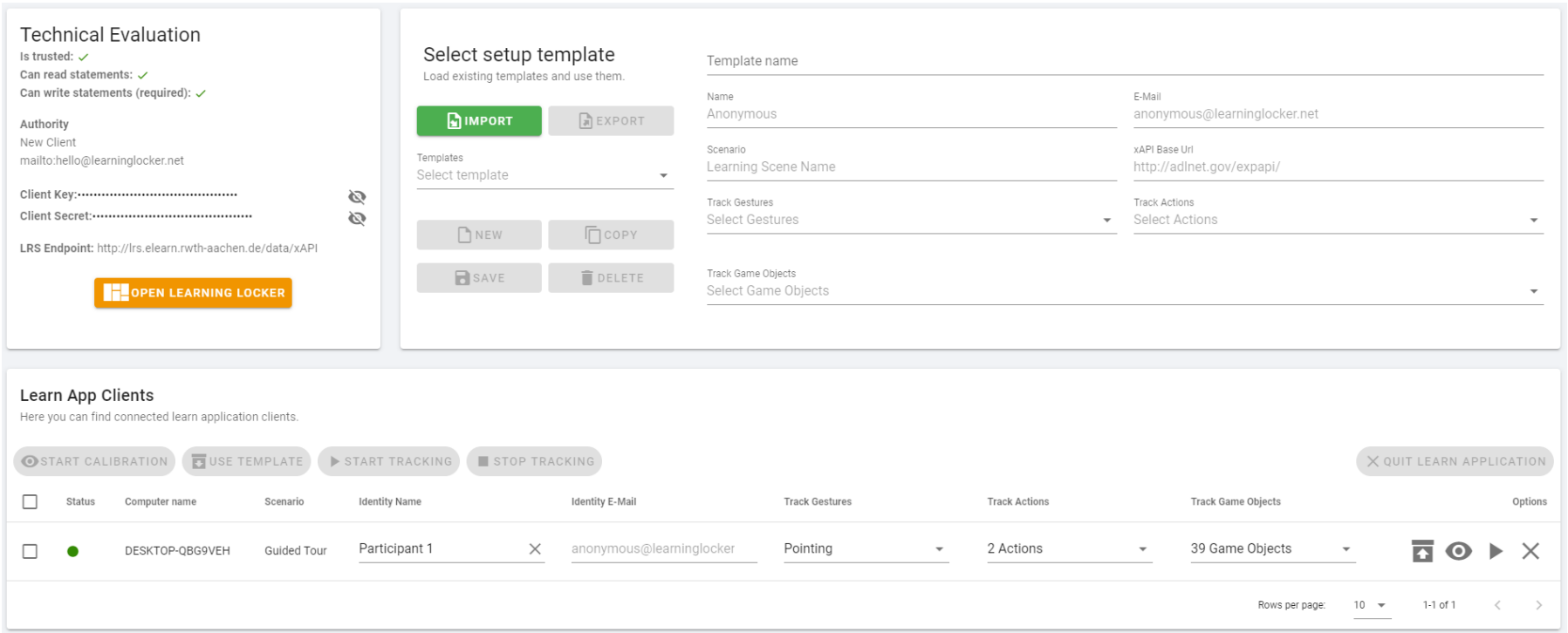


Figure D.14: Researcher Panel during the Guided Tour

Appendix E Questionnaire Results

To identify personas five questionnaires were conducted via *LimeSurvey*[1] in two languages (german and english). The participant choosed their favored language answered the surveys at any place in their favored language online. For the evaluation only completed questionnaires were used. German answered were translated into english and the evaluation was done in english.
In the questionnaires for pupils, students and developer more people have started the questionnaires and didn't finish them. For those questionnaires, I've tried to find the reasons for not finishing the survey ("drop analysis"). Each questionnaire had 15 – 16 questions about anonymous personal information and relation to Computer Science, Computer Graphics and Virtual Reality inspired by Jerald's [43] persona design. Table E.2 provides an overview of all questions. In the upper part have questions which were applied for each persona. The lower part has the questions which are restricted to individual personas (marked by small number). The followed sections provides the meta information to each questionnaire. Table E.1 has an overview over the conducted questionnaires. A complete evaluation of the surveys can be found on the submitted in the digital appendix on the SD card of this work. The interested reader can get more information about the persona evaluation in the extended appendix. To get access to this material they shall contact the author or LuFG i9.

[1] LimeSurvey: `https://www.limesurvey.org/` (visited on 11/25/2020)

Persona	Timespan	Completed	Notice
Pupil	16. June 2020 - 25. June 2020	33/42 (drop rate is 21.43%)	Many (not all) of following tested pupils are visiting a STEM-subject classes. This explains a big interest for Computer Science or Medicine. The results were used to create a realistic model (persona) of the user "Pupil".
Student	25. June 2020 - 02. July 2020	10/19 (drop rate is 47.37%)	-
Teacher	25. June 2020 - 02. July 2020	3/7 (drop rate is 57.14%)	-
Researcher	25. June 2020 - 02. July 2020	3/3 (drop rate is 0.00%)	-
Developer	25. June 2020 - 02. July 2020	11/15 (drop rate is 26,67%)	All developers are software developers (not VR developers).

Table E.1: Overview of the conducted questionnaires

Generic Questions
How old are you? (Type a number)
What are your hobbies? (Call up to three hobbies)
Do you have special competencies? (Call up to three competencies).
Do you have any trait you want to tell?
Do you have physical problems, issues or disease? (Call up to five issues)
Do you have currently any worries?
What do you want to become later?
How high would you rate your knowledge?
How would you rate following topics?
Which VR system you have tried already?
What is your dream VR system?
What do you think about the future of VR?
Do you have access to VR hardware?
Are you doing some activities with VR?
Special Case Questions
What class do you visit?[1]
What do you study?[2]
In which semester are you?[2]
How many years teaching experience do you have?[3]
Has teacher/lecturer been your dream job?[3] If so, do you have other ambitions? if not: what did you originally want to be?
What are your research fields?[4]
How many years research experience do you have?[4]
Has researcher been your dream job?[4] If so, do you have other ambitions? if not: what did you originally want to be?
Which kinds of software are you developing?[5]
How many years developer experience do you have?[5]
Has developer been your dream job?[5] If so, do you have other ambitions? if not: what did you originally want to be?

1 Only for pupils, 2 Only for students, 3 Only for teachers, 4 Only for researchers, 5 Only for developers

Table E.2: Persona questionnaire questions

Appendix F Digital Appendix

The digital appendix is submitted on a SD card. To get access to this material they shall contact the author or LuFG i9.

Eidesstattliche Versicherung

Görzen, Sergej	375348
Name, Vorname	Matrikelnummer (freiwillige Angabe)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/ Masterarbeit* mit dem Titel

Technology-Enhanced Learning of Computer Graphics Rendering Pipeline in Virtual Reality

selbstständig und ohne unzulässige Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Aachen , November 16, 2021	
Ort, Datum	Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

§ 156 StGB: Falsche Versicherung an Eides Statt
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt
(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

Aachen , November 16, 2021	
Ort, Datum	Unterschrift

Acknowledgement

First and foremost, I am grateful to Prof. Dr.-Ing. Ulrik Schroeder for letting me write my Master Thesis at his chair and Prof. Dr. Leif Kobbelt for being my second reviewer.
Next, I want to express my sincere gratitude to my supervisor Birte Heinemann, for her diverse expert knowledge, inspirations, support and guidance during the whole process of this thesis.
Great thanks to my second supervisor Patric Schmitz for his great ideas and hints and for being my contact person for questions about Computer Graphics and Virtual Reality.
Further, I want to thank Matthias Ehlenz for his expert knowledge in didactics and Learning Technologies.
Moreover, I want to thank all participants of my questionnaires and pilot study.
Special thanks to Anne Loos and Roland Müller for proofreading my thesis.
Last but not least, I want to thank my family, my friends and especially my wife Diana for their great support over the last few months!