NTNU
Norwegian University of
Science and Technology

VRterra

Author(s)

Mårten Nordheim
Nichlas Severinsen

Bachelor in Game Programmering
20 ECTS
Department of Computer Science and Media Technology
Norwegian University of Science and Technology,

18.05.2016

Supervisor(s) Simon McCallum

Sammendrag av Bacheloroppgaven

Tittel:	**VRterra**
Dato:	18.05.2016
Deltakere:	Mårten Nordheim Nichlas Severinsen
Veiledere:	Simon McCallum
Oppdragsgiver:	Norwegian University of Science and Technology
Kontaktperson:	Simon McCallum, simon.mccallum@ntnu.no, 95432149
Nøkkelord:	VR, Terreng, Sandkasse
Antall sider:	108
Antall vedlegg:	12
Tilgjengelighet:	Åpen

Sammendrag: VRterra er en samling av programmer og verktøy for avlesing av og prosjektering av terreng på en sandkasse, pluss et Virtual Reality (VR) spill hvor nevnt terreng brukes. På denne måten kan én person styre terrenget og omgivelsene til personen som er i Virtual Reality.

Summary of Graduate Project

Title:	**VRterra**
Date:	18.05.2016
Authors:	Mårten Nordheim Nichlas Severinsen
Supervisor:	Simon McCallum
Employer:	Norwegian University of Science and Technology
Contact Person:	Simon McCallum, simon.mccallum@ntnu.no, 95432149
Keywords:	VR, Terrain, Sandbox
Pages:	108
Attachments:	12
Availability:	Open

Abstract:

VRterra is a collection of programs and tools for reading from and projecting onto a sandbox, plus a Virtual Reality (VR) game where said terrain is used in some way. This way one person can control the terrain and environment for the person in Virtual Reality.

Preface

We would like to thank:

- Simon McCallum for lending his ideas, supervising us and pointing us in directions when we were lost, as well as helping us with the report and the LaTeX in it.
- Johannes Hovland for setting up a plain VR project in Unity 5 so we could do the VR demo and testing.
- Vitensenteret for making two sandbox-setups, one for them and one for NTNU in Gjøvik.

Presentation of project employer:

Simon McCallum has been our course supervisor for three years, and he is part product owner of VRterra, he holds 20% of the intellectual property of the project, together with us. As stated above he is our supervisor for this project as well.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Project Description

1.1.1 Background

The project involves reading data from a real sandbox using a Kinect, processing that data on a computer and projecting the processed data on said sandbox using a projector. It is intended for this data to be used in a virtual reality game, so the project simultaneously allows others to pull the processed data to use in other applications. As such this project can be viewed as the basis of other projects, with us specifically having games in mind.

There have been several independent projects that read in terrain data from a Sandbox, most notably is the SARndbox by Oliver Kreylos [1] – a GPL licensed free software that reads in terrain from a Sandbox using a Kinect and projects a colored heightmap back onto the sand, while also simulating liquids like water and lava. According to Kreylos, the SARndbox was inspired by a video from a group of Czech researchers showing an interactive augmented reality sandbox. Kreylos has a video [2] showcasing what the end result looks like.

There is also an early stage game called Project Mimicry [3], that seems to read terrain from a Sandbox and use it in the game. Several other institutions have implemented their own setups using Kreylos work.

1.1.2 Goal

The goal of the project is to create a technical demo that uses real-time terrain input read in from a physical sandbox in the real world and then use this terrain data as a game element in a game, preferably a VR game. The terrain and positions of players in the game will be projected out onto the sand again, so the sand will be colored and people can see player positions. Simon also wanted us to make an API and a way to get the terrain data from a server, over the network, to a client. In addition, he also wanted our project to be compatible with current SARndbox setups, since that way other SARndbox setups could run our project out of the box on their own setups. This mean that the project would have to compile for Linux Mint with the hardware that Oliver Kreylos has recommended on the project's website [4].

1.2 Audience

1.2.1 Product Audience

The product audience is mainly for exhibitions and institutions, especially those who already have SARndbox already set up, but also for those who have the ability to set SARndbox, or our project, up. The reason we chose to list mainly exhibitions and institutions is because this project is more of a technical demonstration, rather than an actual game. It is also worth to note that most regular people usually do not just build a sandbox with a projector and a kinect in their living room.

On the other side, if the audience has access to a server-setup running our real-time terrain reading software they can use any platform to download the terrain data read in and use it in a program or a game. By making a server, anybody who has access can use the data read in for any purpose over the network, making it flexible and non-platform dependent. The server does however run alongside the sandbox application which was created for GNU/Linux, as that is what Oliver Kreylos' setup already runs on.

1.2.2 Thesis Audience

We target international computer science communities interested in real-time terrain, games, virtual reality and augmented reality. We are expecting that educated or experienced computer scientists read this, therefore, basic computer science knowledge as well as programming experience with C++ (particularly C++14) is expected. Some knowledge about video games could come in handy too.

1.3 Academic Background

We are Mårten Nordheim and Nichlas Severinsen, both Game Programmers who attended the game programming computer science course on NTNU in Gjøvik, previously Gjøvik University College. Both have played video games for many years. We are educated mainly for game programming in C++, but our course includes many other subjects including, but not limited to; networking, databases, algorithms, operating systems, as well as other programming languages. Do note that both have experience elsewhere from education, work and personal projects.

We have most experience with C++, specifically C++14, but we also have experience in other languages; C, C#, Java, Python. We have been in front of computers ever since we were kids and work on them and computer related subjects both as a hobby, in education, and for our jobs. Nichlas uses GNU/Linux and Mårten uses Microsoft Windows, we are about equally skilled in both.

Neither have any practical experience with big projects like this in any form. We took a theoretical system development course in our second year of university to learn about development theory – methods and such, so we know how to plan, we just do not have any practical experience fulfilling plans.

1.4 Roles

Both of us are programmers/developers, neither are designers/artists, and that type of work will be done fast, only well enough to fulfill the requirement, wasting no time. Crucial design decisions are done in plural, and work is distributed to whoever wants to or can solve it. Deciding to work on another task either because one is stuck or because it is more crucial that that part of the system gets done, is perfectly fine.

Simon McCallum is our supervisor; supervising the project, giving feedback for the project and our thesis. The intellectual property of the project is split as such:

- Simon: 20%
- Mårten: 40%
- Nichlas: 40%

In the last week of the project Vitensenteret finished building two sandbox-setups, we did not get to test these however as there were too little time.

1.5 Development

1.5.1 Software Development Methodology

When we made the project plan we chose to use an agile software development plan called Scrum. This way we could iterate over tasks every week, and split up the software part of the project into several “sprints”. Sprints are development events lasting usually between one and three weeks, where the focus of the sprint is to further the development of software. Originally, we chose to go with 13 sprints where each were one week long. This later changed to 16 sprints to give us more time.

At the end of every Friday in every sprint, unless one of us was unavailable, we would have a scrum review (what had been done), a scrum retrospective (what we were satisfied with and what we were not satisfied with), and sprint planning for the next week where we would set the goals for the next sprint. This allowed us to see the progress we were doing every week, but also reschedule and plan appropriately for the next one – this was not always easy, as some goals relied on unknown factors like if we would get a Sandbox or VR hardware.

1.5.2 Schedule

For the project plan we made a gantt chart along with six milestones to guide our progress in this project. This plan turned out to be very optimistic in regards to our estimations and we soon realized the project had been overscoped. On top of this we ended up making unrelated changes during the project due of external input and reprioritization.

The original gantt chart can be found on page 57, with an edited and updated version that shows how the project actually ran on page 59.

1.6 Document Structure

The document structure of this thesis is split into nine main chapters:

- Introduction: Project introduction with background of the project and us, including project intro to planning and development.
- Requirements and Design: Design of the physical setup, the virtual reality game for both the planned project and final setup; what we ended up with.
- Technical Design: Detailed walk-through of the modules and the system architecture.
- Development Process: Description of tools and technologies used, and our workflow.
- Implementation: How we actually went about implementing the project, with a focus on interesting challenges we went through.
- Deployment: Packaging, publishing and deployment of our software; what we did and alternatives.
- Tests and VR Testing: Tests written, VR Testing.
- Discussion: Discussion of our results, workflow, and further development.
- Conclusion: Evaluation of the project.

Appendices come after the last chapter.

1.7 Terminology

AR Augmented Reality. Computer augmented reality.

VR Virtual Reality. Computer-generated environment or a world which can be interacted with using special equipment (in this case a headset).

Heightmap Virtual data structure for storing surface elevation (terrain) data. Used loosely in this thesis when talking about the data structure we use to store the terrain.

JSON JavaScript Object Notation. Data interchange format based on JavaScript.

API Application Programming Interface.

GUI Graphical User Interface. On-screen interface in which the user can interact with.

UML Unified Modeling Language. General purpose modeling language standard.

WBS Work Breakdown Structure. A UML based hierarchical diagram of work to be done.

Protobuf Protocol Buffers. "A language-neutral, platform-neutral extensible mechanism for serializing structured data."[5]

2 Requirements and Design

2.1 Original Design

This section includes a description for how we imagined and planned the project in the beginning of our thesis, similar to the project plan in Appendix A. There will be a final design section describing what changes were made and what was cut out and dropped at the end of this chapter.

2.1.1 Physical Setup

The physical setup of our project would consists of:

1. A sandbox
2. A 3D depth-camera (kinect)
3. A projector
4. A computer to connect all of the above
5. Optionally screen and/or several other computers.
6. VR hardware

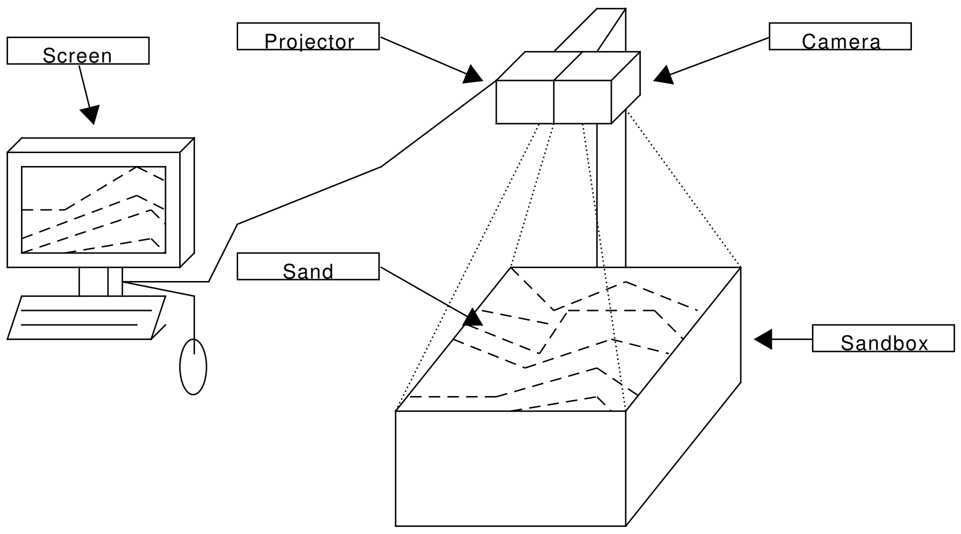


Figure 1: Visualization of the physical setup. VR hardware not visualized. Image from the project plan in Appendix A

The 3D depth-camera reads in the terrain from the sandbox which is then sent to a computer. The computer would be running a sanbox projection program (a list of the different programs can be found in Section 2.1.2) which uses the projector to project back onto the sand. In SARndbox the use case for this was to simulate liquids on terrain, for VRterra we will project player locations on a shaded terrain.

2.1.2 Virtual Setup

The original virtual setup consisted of several programs and libraries

1. Terrain Library
2. Network Library
3. Engine
4. Terrain Editor
5. Sandbox
6. Game

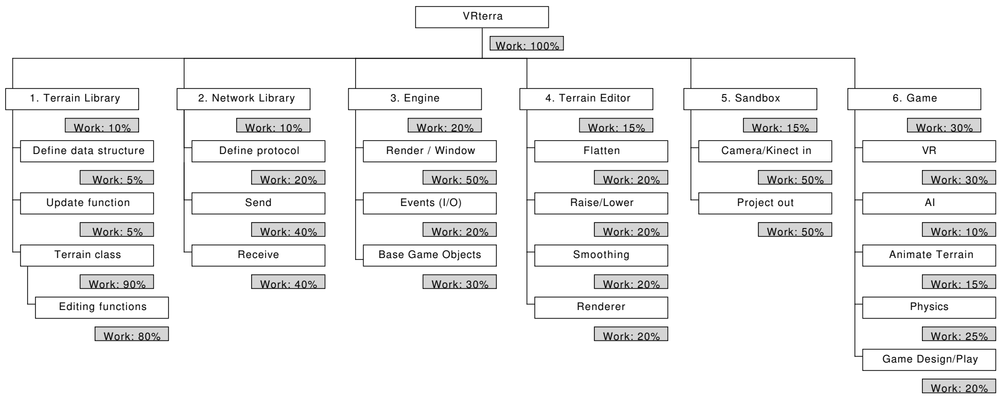


Figure 2: WBS; Work Breakdown Structure. Image from the project plan in Appendix A

Above is our work breakdown structure which shows how we split and estimated the percentage of work required for the different parts.

The terrain library would consist of a data representation of terrain, a heightmap, and functions related to terrain and terrain editing. The networking library would have a server and a client that could communicate data between them; the terrain heightmap and positions of the player. The engine is meant to represent our "Base" program, of which all other programs relied on for rendering in 3D on screen. The Sandbox, which was later renamed and specified to projection program, would both read in the terrain from the sandbox using the 3D camera, and project a colored version of the terrain – with players – back onto the sand. The game acts as a client that receives terrain data, it is also supposed to have VR capabilities.

2.2 Game Design

In this section we will discuss some different game design aspects to consider when developing a game using our setup, as well as list some concrete game ideas.

2.2.1 Real-time vs. Continuous vs. Once

Whether or not the game is real-time, continuous, or once read in needs to be considered. Currently there is no real smoothing, in neither game nor projector, except for frame averaging. For real time it would be a good idea to smooth the transitioning out between terrain read in that differ too much. A physics based game, for example a ball game, fits well for real time terrain. Continuous in this setting means not real-time but for example every 10th second, the terrain would still need to transition. Reading the terrain in once means you do not have to do transitioning, at the cost of not updating terrain. A competitive game where users create the terrain once, for example a first person shooter or maybe even a MOBA, could work well for read-once terrain.

2.2.2 Interface; Screen vs. VR vs. Sand

Depending on the game type and genre one could mix and match what interfaces to use and not to use. For example, in a car driving game VR would work well; you sit in a seat but can look around while driving, while the aforementioned physics based ball game could possibly use an AR interface; displaying on the sand to show the balls bouncing - which would make the game more interactive than seeing the balls on a screen. A competitive game like the suggested first person shooter game would likely have to be on a normal screen with keyboard and mouse, however, as competitive games traditionally are.

2.2.3 Single-player vs. Multi-player

The server we developed during this project has capacity for sending terrain to many clients at the same time, so creating a multi-player game would definitely be possible; having multiple people driving around in cars on an ever-changing terrain that some other players were controlling, trying to complete objectives like jumping and doing tricks.

2.2.4 Concept Idea: Multi-player Offroad Game

While visiting and talking to Hamar Game Collective (in Hamar, Norway), see appendix F.1, we collectively came to the conclusion that a good game design idea for our setup would be to create a car driving game where players could drive cars in VR while the terrain was slowly animated between the different data read in by the 3D camera. The game element would consist of making jumps in the sand with the sandbox in order to complete objectives by jumping with the car; like reaching a certain height or collecting floating objects in the air.

2.3 Final Setup and Design

Many things changed from what we originally planned while we were doing this project. There were some major changes, including switching game engine, and many minor changes. In this section we list the final setup and design that we ended up with, with emphasis on the actual changes.

2.3.1 Physical Setup

We did not really obtained a sandbox to experiment with until very late in the project, so for some time we made do using a table with a cone-hat and some toys on top. We only tested VR once, in the last week before the deadline, by having Johannes create a basic flat-terrain VR setup in unity using the Oculus Rift, and then we would load our terrain in with a script, once. The VR hardware was never moved and set up with the physical setup. Therefore, the remaining physical setup consisted of these parts:

1. A table with a makeshift sandbox
2. A 3D Camera (The kinect)
3. A microphone stand holding the kinect over the table
4. A roof-mounted projector projecting onto the table
5. A computer connecting all of the electronic parts; monitor included.

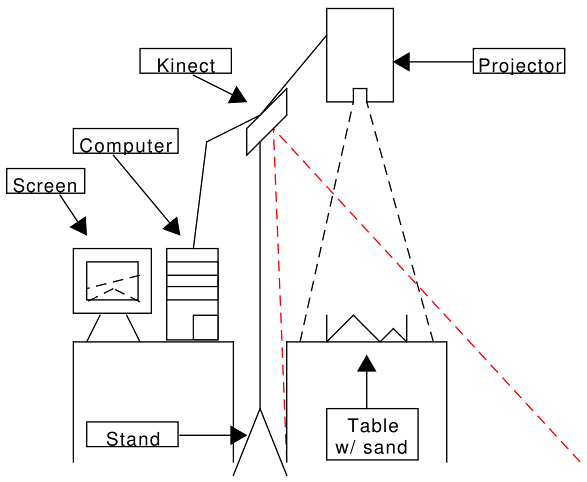


Figure 3: Final visualization of the physical setup.

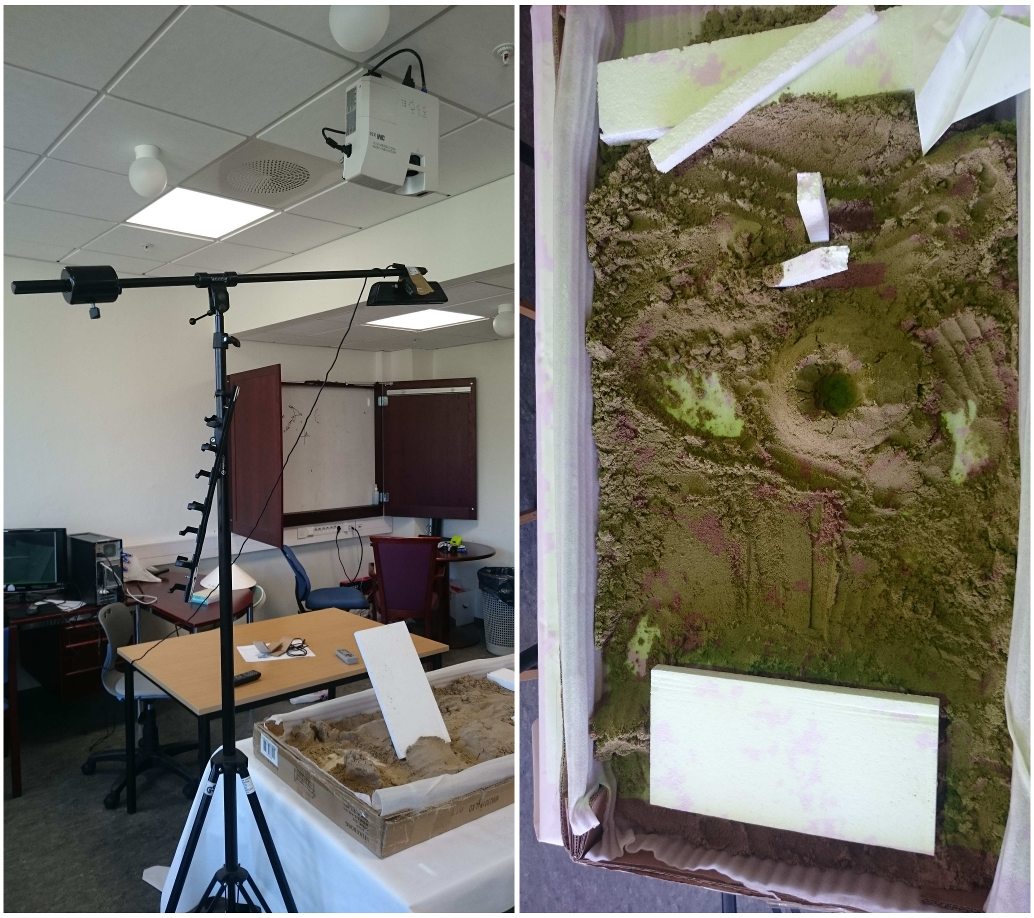


Figure 4: And here is how it looks like in real life

2.3.2 Virtual Setup

After handing in our project plan we set to work creating our own game/graphics engine, however as we progressed we were recommended to use an existing engine instead. He suggested we use Torque 3D, but after both of us failed compiling it on our machines a search for an alternative started and we ended up with Irrlicht. An evaluation of the other engines we looked at can be found in Table 2 Additionally, in order to fill the VR requirement, we modified a small Unity project called CarSimUnity [6]. This project is now a car driving VR game where the player sits inside the car and drives on terrain gotten from the server. Early in the project we started creating a "terrain editor" which would serve as our terrain until we got a kinect to work with, we decided on dropping this completely from the project after acquiring the kinect. In the end the final virtual setup consists of these parts:

1. Terrain Library
2. Network Library
   - Server
   - Client
3. FPS Demo/Game

    NoKinect
    Version Using Kinect

4. Unity VR Game
5. Kinect API
6. C# Download Terrain Test/Example

3 Technical Design

3.1 Technology

VRterra is written in C++14, but we did also create a C# program that could get the terrain data from a server via ASIO, TCP – this way others can set up our server software and build an entirely new game while using our server software to get the terrain read in from the sandbox. Our game-demo, proof of concept, was made using Unity for the sake of simplicity; we had to get some sort of demo up and running that also was VR capable – which unity very easily made available, in addition, we will later talk about how the project can be used later and one of the examples there is that other students can look at this Unity game as an example and/or create their own game with or without it.

3.1.1 Libraries

Here is a list of the libraries we used:

Engine; Irrlicht

For rendering and game components we are using an open source game engine called Irrlicht. We did plan on making our own game engine using SDL and OpenGL, but we ended up with Irrlicht because making our own engine took too much time, and introduced many new issues and user stories. Some of the following libraries were written for the engine we initially started working on. After making the switch we then moved the libraries we needed to our new Irrlicht version and continued improving them.

Libfreenect and Libusb

For reading in data from the kinect we have created our own API on top of a library called libfreenect, which is an open source library that communicates and gets data from the kinect. In order for libfreenect to work we also had to include libusb-1.0.

ENet and Google Protocol Buffers

For networking we used ENet with Google Protocol Buffers to communicate between our own applications. ENet is the networking library while Protocol Buffers is used to structure packets which is between the clients and the server. ENet is C++-specific and requires the specific library to function. For the networking API it was crucial that it was easy and possible to connect from potentially any language, and as such we could not use ENet for this as it is a thin layer on top of UDP to provide reliability and in-order delivery of packets. Forcing others to reimplement this layer in their language of choice was not an option, so we opted to use TCP for the API.

ASIO

We used the ASIO library for TCP. To easily structure the protocol over TCP we decided to use JSON-encoded messages.

JSON for Modern C++

For JSON parsing we are using a library called "JSON for Modern C++" which we used for our applications' settings files as well as the TCP api.

Microsoft Guidelines Support Library

We used Microsoft's Guidelines Support Library for its convenient features. It lets us use gsl::span to wrap containers with a generic interface. As well as macros for assertions to ensure that if we end up in a non-recoverable state the program will not continue executing.

Google Test

Unit tests are written using the Google testing framework. For details on our different tests see chapter 7.

3.2 System Architecture

The final self-made system architecture consists of many small and larger parts. We will go through them with technical detail, show some of the code, and visualize the system architecture with the most important parts.

3.2.1 Overview

1. Libraries / API
   - Terrain Library
   - Networking Library
   - Kinect API
   - C# API
2. Programs
   - Projector program (Irrlicht)
   - FPS Demo (Irrlicht)
   - VR Game Demo (Unity)

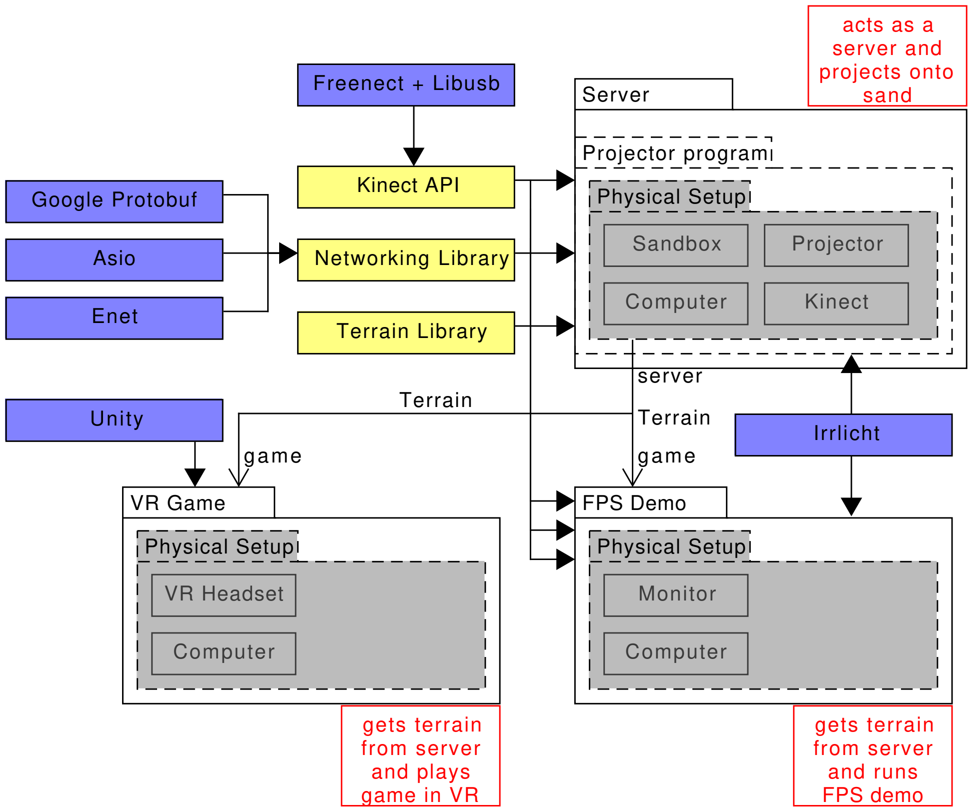


Figure 5: Simple visualization of the system architecture. Yellow boxes are libraries we have made. Blue boxes are third-party libraries and modules. Short description of program in red text.

3.2.2 Terrain Library

The terrain library is a C++14 written class which main part consists of a 1D heightmap while the surrounding parts are functions for manipulating said heightmap. What is interesting to note is that the terrain itself is a 2D heightmap, so when storing it in our terrain library we have to do conversions from 2D to 1D and back, meaning we have to store either the width of the original heightmap. We can then take a X and Y coordinate and convert it into an index using

$$\text{index} = x + y \times \text{width}$$

Similarily, in order to convert from 1D to 2D we use

$$x = \text{index} \ \% \ \text{width}$$

$$y = \lfloor \text{index} \ / \ \text{width} \rfloor$$

These conversions were found on StackExchange [7].

Because the terrain is read and written to from many threads on the server it was required that reading and writing to the heightmap would be thread-safe. While the sandbox is real-time it is not vital that the sandbox's frame rate stays high or consistent which could have let us use mutexes or a read-write lock to synchronize access. We did however end up implementing it with the copy-on-write strategy where every read-operation takes a reference to the heightmap and uses that until the operation completes. The write-operations takes a reference of the heightmap, works on it and then exchanges the original atomically once the operation completes. This way every on-going read-operation can keep reading the original uninterrupted while the write-operation does its own work. There is only ever 1 write operation at a time so no information is lost by consecutive writes since there will never be another 'writer' which starts writing after the initial read.

The terrain class also lets us register callback functions to be called whenever a value in the heighmap has changed. This made it easier to keep different parts of the application synchronized when a change was made from any other part of the application. The callback functionality was implemented using std::function which we store in a map (dictionary) and invoke with relevant variables when a change has been made.

3.2.3 Networking Library

This section will talk about the parts relevant to the networking in our project. It is separated into two parts: the components and the protocols. The Components section will talk about the various classes which make up the networking code aspect of our project, while the Protocol section will talk about the two protocols we defined and used in the project.

Components

In our networking library we have the overarching Networking-class, the Server and TcpServer classes and the Client class.

Networking

We chose ENet to use in our networking, you can see a comparison of a few network libraries we were looking at in section 5.3.1, and wrote the Network class as a wrapper around the functions we needed. Other networking-related classes contains an instance of the Network class and uses it to connect to a server or receive connections from multiple clients. To make it easy to work with the packets we decided to use Google's protocol buffers to easily serialize and deserialize the data in a structured way. More information about how we used protocol buffers on page 15.

Client

The Client class was created to make it easier to connect to a server and to retrieve changes made to the terrain. It also only needs to handle a single connection because it only ever connects to one server at a time. The class requires that you call its update function to handle any received packets. The update-function takes a pointer to the terrain, which it will alter based on incoming data from the server.

Server

The Server starts its own thread and then runs in a loop until it is told to shut down. On each iteration of the loop it looks for and handles any incoming packages, and if any changes have been made to the terrain it will broadcast the terrain to all currently connected clients. The broadcast has a maximum rate at which it broadcasts the terrain which, by default, is set to 5 seconds but it can be changed using the configuration file. Currently we are transmitting the entire terrain every time, but in an earlier version it would only send the changes (deltas) that had occurred. Why we made it and why it is currently not in the project can be found in section 5.3.2.

NetUtils

Although it is not a class, we feel it deserves mentioning. NetUtils is a collection of free functions which is used by the Network, Client and Server class to serialize protobuf messages into bytes before transferring it, extracting protobuf messages from packets and applying information from the appropriate protobuf messages on the heightmap. It also has a couple of extra, useful, functions to compare two peers to see if they are the same and to get the string of an IP-address from a given address from within ENet. The latter is only used for printing who connects and who disconnects.

TcpServer

The TCP server came about as the result of a requirement to have an easily accessible API. It is an inner class of Server and can be started by calling the Server class's startTcpServer-function. This starts a thread which listens for and accepts a new connection. Every time a new connection is received a new thread is started which handles the incoming connection from that point on and the listening thread goes back to listening for a new connection. As we mentioned earlier we use protocol buffers for the Network class, which means we also use it for the Client and Server classes. Even though protocol buffers was a great help in making it easy to efficiently serialize and deserialize data it requires that the language is supported by the proto compiler, or that a plugin for the proto compiler has been developed to support the language, and to include the protocol buffer library in compilation. In an effort to keep this API easy to use we decided to use JSON to encode the data we need to send. This choice has other implications which will be covered on page 16. A second requirement for the TCP interface was that developers should have the opportunity to add a marker and set its position. This marker would then be drawn at its chosen position on the output screen/projection surface. The cross-thread communication was achieved by using a Singleton, called MarkerManager, which contained all of the markers and their position.

Protocol

Our first protocol used protocol buffers and because of the requirement mentioned earlier to have easy access to the terrain data we decided that we needed to implement another protocol for the TCP communication using JSON. This left us in the position where we are now using protocol buffers for communication over ENet and JSON-encoded messages over TCP.

Protocol Buffers

We chose to use protocol buffers since we thought it would make for a good learning experience as neither of us had used the library before, but we had heard good things about it, and thought it would be a good fit for the project.

Protobuf required us to write a proto-file which defines messages. These messages define structure of packets in our protocol. One of the messages we defined in our project is shown below.

```
message FullHeightmap {
  repeated float heightmap = 1 [packed = true];
  required uint32 columns = 2;
  required uint32 rows = 3;
}
```

Figure 6: One of our protobuf messages, contains a repeated field which is the values in the heightmap, and columns and rows which signify the width and height of the heightmap respectively.

This proto-file then had to be compiled with the proto compiler into C++ code which is then included and compiled into our program. This gave us a very nice interface for dealing with packets specific to our project as opposed to defining byte offsets or parsing XML/JSON and using generic functions to get access to the content we needed.

```
// "heightmap" is the FullHeightmap message shown earlier
terrain->setNewSize(heightmap->columns(), heightmap->rows());
// --- SNIP ---
std::vector<float> buffer(size);
// note the begin() and end() functions
// on the heightmap property for easy ranged access
std::copy(heightmap->heightmap().begin(),
    heightmap->heightmap().end(), buffer.begin());
// --- SNIP ---
```

Figure 7: One way we used the message shown in figure 6.

The only issue we encountered while using protobuf was that it was impossible to know what message was stored in an incoming packet. You can read about how this issue was solved in section 5.3.3 on page 34.

JSON

Unlike protocol buffers which has to be compiled, JSON is designed to be much more generic and does not compile schemas into classes. JSON is also human-readable and transmits clear text while protobuf encodes data to binary. In short, this means that the JSON messages are larger than the protobuf messages even though we are sending the same information.

```
{
  "columns":640,
  "flag":2,
  "heightmap":[125.0,126.0,...,500.0],
  "rows":480
}
```

Figure 8: An example of a JSON-encoded message which has been 'prettified' for readability. Do note that there would usually be (columns * rows) values in the heightmap array, which has been mostly omitted.

To identify which type of packet every packet has a "flag"-field which corresponds to a value in an enum in the project. After checking the flag the rest of the packet is assumed to be correct.

The choice to use JSON for networking was mostly affected by the fact that it made it very easy to parse the message in many other languages as there are many libraries for parsing from and serializing to JSON. The fact that we were already using JSON for our configuration files made this choice even easier.

However, because we are sending this data using TCP we do not have native delimiters around the JSON-message. This means we had to find a way to signify the start and end of a JSON message. Read about how we solved this in section 5.3.4.

3.2.4 Kinect API

The kinect API is a small, simple, C++14 class that inherits from the freenect driver. The freenect driver is a GPL licensed driver for the Kinect, maintained by the openkinect community.

3.2.5 C# Library

The C# library was initially written as a rough way to test the TCP-api on the server. It evolved as time passed to be easier to use and hide away the JSON packet creation and so on. It also de-evolved from .NET framework 4.5 to .NET framework 2.0 to be able to use it in Unity which is the library's main use case. The library has a dependency on Json.NET from Newtonsoft to serialize and deserialize JSON messages. Since the library uses TCP a decision was made to have it run a thread dedicated to listening for incoming data and handling it. This allowed us to avoid blocking while waiting for the rest of the JSON message if it had not been received in its entirety. To further avoid blocking during communication you do not directly send any messages to the server, but rather call functions which will queue messages to be sent by the TCP client thread as soon as it can.

3.2.6 Projector program

The projector program is usually run on the computer plugged into the projector and the kinect. The projector is simultaneously running the server we wrote about in section 3.2.3. The projector program is simple, it gets the terrain read in by the kinect, and it uses Irrlicht to project a colored/textured version of the terrain back onto the sand. Currently one can also send the projector server program positions to move GUI markers in 2D on the projection, to display player positions and such.

3.2.7 FPS Demo

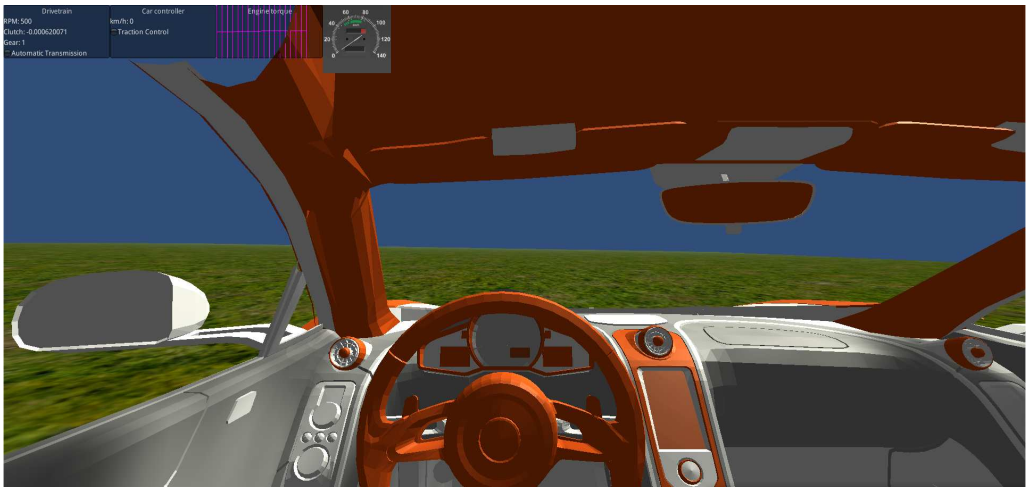
The FPS demo, like the projector program, was made using Irrlicht version 1.8.3. The FPS Demo is only a camera floating in space, together with the terrain displayed using Irrlicht and irrlicht terrain objects. The user can fly around and change to wireframe-mode while the terrain is being updated in real-time.

3.2.8 VR Game Demo

This section includes two subsections; a description of the VR game demo and how it was developed.

Description

Our VR game demo includes a car on a terrain that is loaded from the kinect, through the networking library in the server, via the C# API and into our game. You can drive around on this terrain with the car, and if you attach a VR headset you will be able to sit inside the car and look around while driving.



Drivetrain
RPM: 500
Clutch: -0.000620071
Gear: 1
Automatic Transmission

Car controller
km/h: 0
Traction Control

Figure 9: Our edited version of CarSimUnity

In the last week of the project this was changed to become a first-person VR Demo instead, meaning no cars. A picture of this can be seen in Figure 27.

3.3 Program Flow

3.3.1 Projector Program Flow

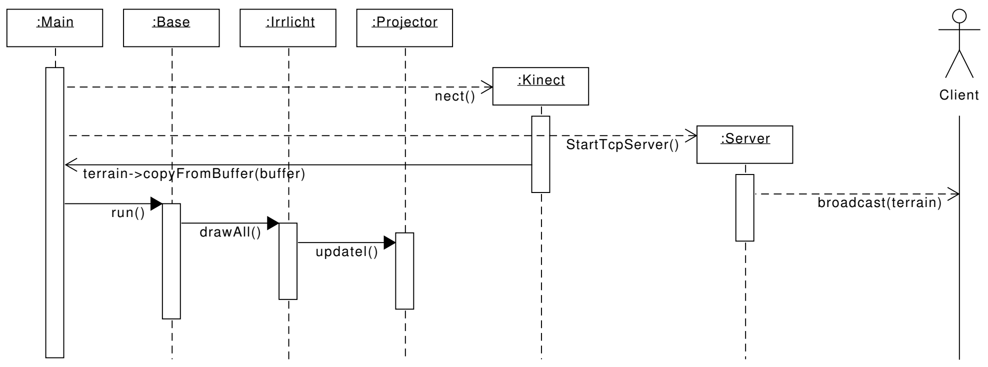


Figure 10: Projector Program Flow.

In the image above, the main projector file will initialize two new threads; Kinect and Server. They all share terrain via pointers. Main will then proceed to run base, which will loop to draw in irrlicht and call on projector specific functions to update.

3.3.2 FPS Demo Program Flow

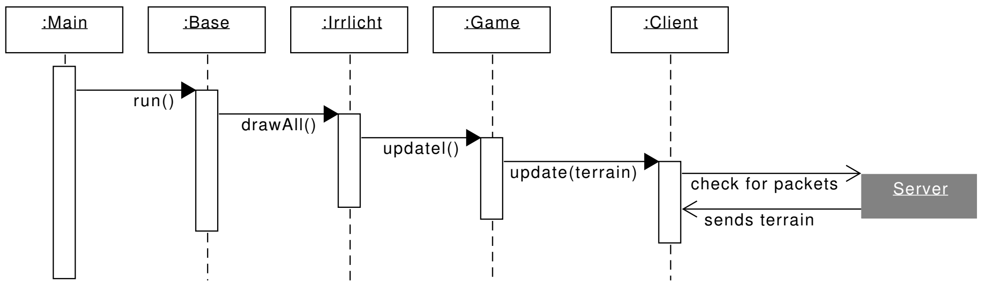


Figure 11: FPS Demo Program Flow.

3.4 Other

This section will talk about the classes and functions that do not fit in in the other sections; Currently we have the Utils library and the Common library.

3.4.1 Utils

Utils is a library we wrote for use in the project. All the functions in this library are free functions, meaning they do not belong to a class, and most functions are header-only, meaning their implementation is located in the same file as their definition. Putting it in the header has the drawback the any change leads to recompilation of every file which includes this header but this rarely happens as these header-only functions rarely require any changes.

Config

Early on we realized that we would need to have an easy way to edit certain variables without recompiling the entire application. This lead us to write the Config-class, which is the only class in the Utils library. Before we implemented it we had to decide on the format for the configuration files. The choice stood between .ini files and .JSON files. While looking for ini-parsers we were unable to find any libraries which is as easy to use (during inclusion, compilation and actual usage) as JSON parsers we were already familiar with which lead us to using JSON.

Here is an example of how the configuration files look like:

```
{
    "WindowTitle": "VRterra",
    "WindowWidth": 640,
    "WindowHeight": 480,
    "Fullscreen": false,
    "Vsync" : false,
    "MaxFramerate": 60,
    "VertexShader": "res/shaders/colorshader.vs",
    "FragmentShader": "res/shaders/colorshader.fs",
    "HostAddress": "127.0.0.1"
}
```

Figure 12: An example of a VRterra config file.

The example above shows the configuration file of one of our applications. It defines the name of the window, the resolution, whether or not to start up in fullscreen and so on. Due to the lack of GUI we load the IP to connect to from the configuration file, this option is the HostAddress option, which can be seen on the next-to-last line the the example above.

To use the Config class all you have to do is load a file, and then start accessing the options in the file. The function to access options in the file is a template function which lets you define which type the value belonging to the option should have. This removes the need to define multiple functions to access different types.

```
Config config = Utils::Config();
config.load("path/to/filename.json");

// Specifies a default return value (127.0.0.1)
auto address = config.getValue<std::string>("HostAddress",
                                             "127.0.0.1");
// The default return value is implicitly
// set to the default value of unsigned (0)
auto width = config.getValue<unsigned>("WindowWidth");
```

Figure 13: Example usage of the Config class. Here we are loading two of the values from figure 12 which showcases its primary usage.

Because some options may not be present in the configuration we gave ourselves the ability to define the default return value which it returns when this occurs. If a default value is not specified then the default value is the result of the constructor belonging to the specified type which takes 0 arguments.

Input/Output

We created a function to read all the content of a file and return it as a string. It contains the minimum amount of error handling, which is to check if the file could be opened properly, and if it cannot it will print an error which was, for this project, always a message that the file could not be found because we had misspelled the filename. We initially had to do this in multiple places so we created a file for I/O functions in our

Utils-library. Currently it is only used by the Config-class to read the JSON configuration files.

Error

Defines a couple of macros which can be used at any place in the program. One of these macros is DBG, short for debug, which, when compiled in debug mode, lets you print any arbitrary text to the screen along with which file and line which the DBG macro was used. It is implemented using stream and is thread-safe so multiple uses of DBG at the same time will not cause the output to be jumbled together.

```
auto status = "Fine";
DBG("Program␣Status:␣" << status);
// path/to/file.cpp (2):
// Program Status: Fine

auto vec = irr::core::vector3di(1, 2, 3);
DBG("x:␣" << vec.X << ",␣y:␣" << vec.Y << ",␣z:␣" << vec.Z);
// path/to/file.cpp (7):
// x: 1, y: 2, z: 3
```

Figure 14: An example of how to use the DBG macro, the comments below each DBG-use is the expected output.

The other macro is FATAL which does the same as DBG but does not stop working outside of debug mode and, as it name implies, should only be called when the program enters a fatal state, meaning there is no reason to continue running. It terminates the application after the output has been printed.

3.4.2 Common

This section describes classes that, similar to Utils, is used in multiple places but do not fit in to the Utils-library. Currently the only classes are Marker and MarkerManager.

Marker

A small structure which contains information about a marker; It has id, 2D-position and a currently unused boolean to see whether or not the marker is enabled.

MarkerManager

MarkerManager is a singleton class which holds and manages markers. Using it you can retrieve a reference to a marker to edit it or create a new marker. Creating a marker is done by the network server when it received a request from a client. Retrieving a reference is done when a marker gets updated or when the draw-loop needs to draw the markers. Because it is accessed from multiple threads simultaneously it is designed to be threadsafe, which is accomplished using a mutex to lock access to the container holding the markers.

4 Development Process

4.1 Working Hours

Our working hours were from 08:00 to around 15:00, Monday to Friday, at our regular working place which for most of the project was at Mustad. At the end of the project, last few weeks, we stayed at the Game Lab to write and test with the sandbox. We wanted to tackle this task like a job with fixed work hours as opposed to working whenever we felt like it from home. There were other factors like when Mårten's bus was leaving, if Nichlas had driving lessons, if we had a class that day, if someone was sick, etc. that would change the start or end of the day, including the total amount of working hours. We also chose to do it because we are both bad at working from home, plus the fact that the equipment was at school. The last few days before the hand-in of the report were public holidays.

4.2 Development Tools

In this section we will go into more depth about the various tools we employed during the project.

4.2.1 Version Control

We were both taught how to use the Git version control system during a second year course, therefore it became natural to choose git as our version control system. There are several alternatives; Bazaar, SRC, Mercurial. There is also subversion(svn) and cvs – but we consider those to be a little bit outdated and hard to deal with; cvs because of no moving or renaming, and svn because of unnecessary permission system. Our existing experience with Git was the main reason for choosing it, which is why we did not choose any other version control system.

The great thing with Git, since it is a version control system, is that multiple developers using git can work on the same file and "merge" their changes together, into a final version of the file. With smaller changes such a merge is usually automatic, but with larger changes the developers themselves need to do the merge manually – though usually using a "mergetool" that shows all current states of the file. With Git you also have "branches", meaning that a developer can take the current state of all files in a "branch" and "branch off" – creating a new branch that branches off of an old branch. The starting branch of most projects is usually called the "master branch".

There are different workflows, how you use branching and merging, but we decided after experience to use a workflow method known as "feature branch". With feature branching a developer will branch off into a specialized branch for a major feature the developer is working on. For example, when we were working on the Kinect API we branched into a new branch called "feature/#9-kinect". The number in the name coincides with the number on the issue of our issue tracker – Taiga.io.

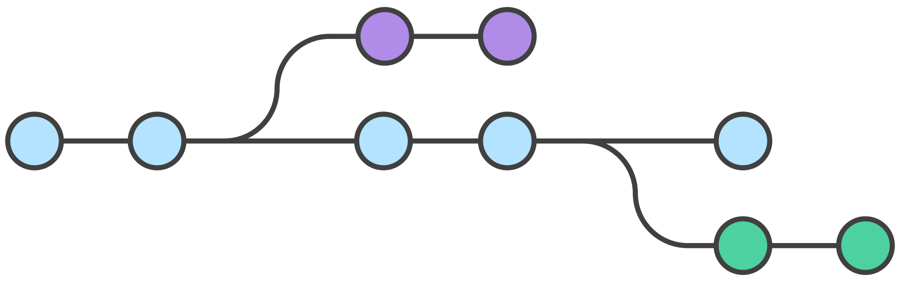


Figure 15: Simple visualization of the feature branch workflow. Image is courtesy of Atlassian

4.2.2 Building

As mentioned, we have used C++14 to develop most of VRterra. We chose to use CMake, a meta-build system, to manage how the project is built, independent from any compiler. With CMake you create configuration files, or scripts, that the CMake program parses to create a makefile. Makefiles are used for automatic building, roughly explained they contain instructions for the compiler – saying things like "take these files and dynamically/statically link them with these libraries", and similar.

For compiling we have been using GCC, Clang, MSVC++, and GCC on Windows with MSYS2/MinGW.

4.2.3 Project Management

Taiga.io [8] is a website hosting Taiga; a Free and Open Source Project Management Tool for Agile software development. Agile is

> [...] a set of principles for software development in which requirements and solutions evolve through collaboration between self-organizing. [9]

With Taiga you can create user stories that are placed in a backlog, plus issues. While our source code was hosted with BitBucket [10] (by Atlassian) under git, our issues and backlog was on Taiga. We did link the two together using webhooks – that way one can do a semantic commit in which the commit message includes a command to the issue tracker (Taiga). An example of this for Taiga is "TG-REF6465 #Closed", which will close the issue with 6465 as a reference. We did however not use semantic commits that much because we found Taiga's semantics to be difficult to remember, especially if you were setting some status other than closed, e.g. "ready-for-test" and "in-progress". Time tracking was done with Toggl [11]. Worklogs were written in a latex document daily and pushed to our own Report git repository.

4.2.4 Coding Environment

Windows

On Windows work was mainly done using Visual Studio 2015 with Jetbrain's Resharper Ultimate for its refactoring capabilities. Pushing and pulling was mostly done using PowerShell with the Posh-Git module for tab-completion although Git Extensions came in handy for staging or reverting individual lines or sections of code. From time to time code was compiled with GCC on MinGW using Ninja to see if the code produced would compile on GCC at all.

GNU/Linux

Most of the work done on GNU/Linux was done in CLion by Jetbrains, although we did use VIM on the machine we were borrowing, which was running GNU/Linux Mint, so we could both SSH into it and edit it from terminal. Debugging was also mostly done in CLion (through GDB), although some bugs like weird segfaults required further use of terminal debugging programs; GDB or LLDB. Git pushing with git command-line.

Irrlicht

Irrlicht [12] is a self-licensed open source game engine written in C++. The Irrlicht license is based on the permissive zlib/libpng license [13]. Irrlicht is small and lightweight, and was relatively easy to start using, though some tasks required extra research in order to be solved. Irrlicht is cross platform and works on both GNU/Linux and Windows.

CMake

CMake is a BSD-licensed cross platform open source build system [14], one can think of it as a level above makefiles. CMake allows you to create configuration files, called CMakeLists.txt, written in CMakes own configuration language. You then run cmake on the top folder with a top CMakeLists, this will generate an advanced makefile which in turn can be used to build the project. Mårten and Nichlas already had some experience using CMake from previous projects.

4.3 Project Workflow

4.3.1 Scrum

As mentioned, we used scrum to manage our project. We have also mentioned that we usually work weekdays from 08:00 to 15:00. A day usually started with a daily scrum, also known as a daily stand-up, where we each tell each other what we did yesterday, what we got stuck with, and what we are going to do today. The purpose of the daily scrum is to keep all team members updated on what is being done, as well as identifying impediments in the project, so as to give an excellent understanding of what has been done and what remains. Our daily scrum was very flexible, often short and precise. In a two-man team like this we would, most of the time, know what the other was was working on, and what needed to be done was listed in the backlog. On the days where we were simply continuing tasks that we were working on the day before we saw fit to skip the daily scrum.

Our day-to-day workflow mainly consisted of coming to school early before 08:00, doing daily scrum, coding and writing until about 12:00 when we had a short 15 minute lunch break and then continuing after that, till about 15:00. We would write worklogs of what we did that day, usually before leaving.

When a sprint ended, usually Fridays, we would do a sprint review, retrospective and plan for the next sprint. The sprint reviews consisted of exactly what was done that sprint, if things not listed in the previously planned sprint goal the week before had been done those things would be listed as well. The sprint retrospective would contain how we felt the sprint had gone, things that were good – and that we should continue with – and things that were bad – things we should definitely stop doing. Most bad things boiled down to being tired. At the end of Fridays we would do a sprint planning for the next week where we wrote down a set of goals, optionally pushed previous goals in front, that should be worked on in the next sprint.

4.3.2 Milestones

We set ourselves some rough milestones when we were planning the project, but after missing the first milestone, "Terrain Editor & VR Game in Alpha", and then changing the engine we knew that the milestones were useless and did not use them after that.

4.4 Development Workflow

We have mentioned how our git workflow was using feature branching, and we have talked about Taiga – our agile project manager, so in this section we will present how we worked more in detail and how we used these tools together.

4.4.1 Source Code

Before anything gets done, a user story needs to be created in Taiga, optionally with additional issues created at the same time – optionally new issues created later. After creation these issues are then assigned to someone based on priority and interest, sometimes just orally between the two of us, and then the person assigned starts working on it. An assignment can be dropped, so the issue goes back to being unassigned, to be picked up later by the same person or by someone else; it all depended on priority and if the person assigned is able to finish the task. When the person assigned starts working on the user story or issue, a new git branch will be made depending on the issue. If the issue builds upon or is relevant to pre-existing branches the assigned can checkout to those, otherwise follow the feature branching workflow. Source code is created, and commits are split up into as many small but descriptive commits as possible – following the git practice recommended by our Professional Programming course which we are taking in parallel to this thesis – and pushed to the remote repo on Bitbucket. Pull requests are made when a feature is considered finished and stable enough for develop, merges can be done locally by the developers as well. Pull requests are then reviewed and either approved or declined by the other team member, declining requires a reason. Code is commented appropriately, but not in excess. The code convention found in appendix D must be followed for code style and naming, plus adding documentation as comments so documentation can be automatically generated using Doxygen. Occasionally someone will use clang-format for automatic formatting of source code to make the formatting look nice.

5 Implementation

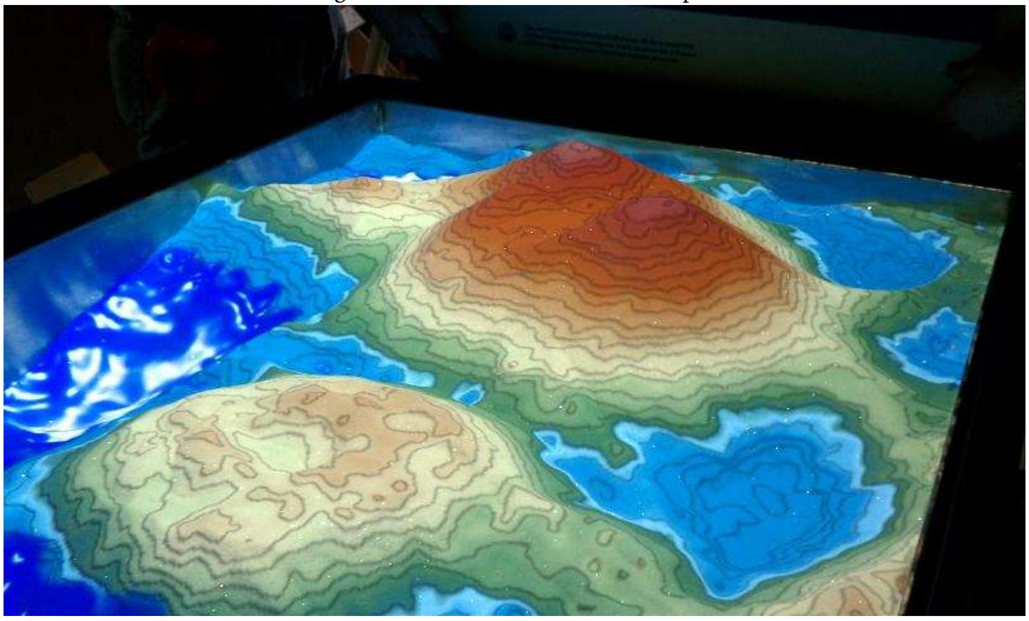
5.1 Terrain

This section describes interesting challenges and problems we had with terrain during the project, how we optionally solved it, or how we think they could have been solved.

5.1.1 Rendering

We had to figure out how to make the shader interpolate from dark to light colors, so that the terrain details would be visible on screen and on the terrain. The SARndbox shader produces output that looks somewhat like this:

Figure 16: SARndbox Shader Example



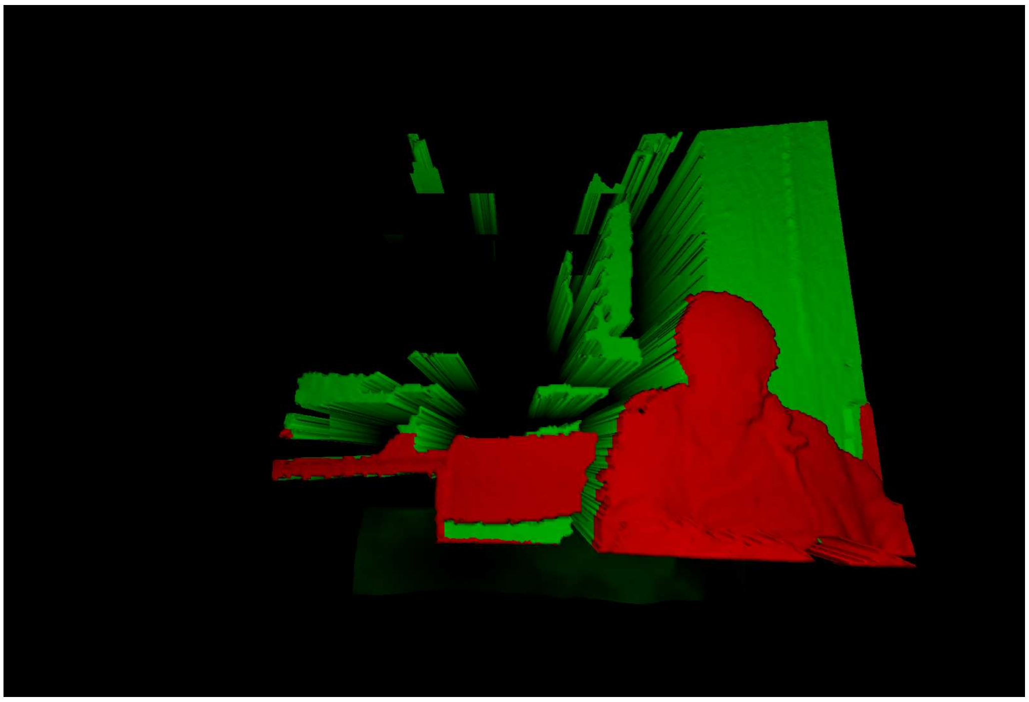


Figure 17: Here you can see an early version of our shader, this is viewed inside the FPS Demo/Game, and is therefore angled and 3D (as opposed to the Projector which shows terrain orthographically)

Originally we thought to write a shader which simply colored different "levels" with different colors. As in; what we deemed to be ground level was green, below ground level was blue and above mountain level was red. This looked fine when there was lighting, but we realized lighting did not make much sense on a sandbox in the real world where there is already lights which create shade on the sand. Thus lighting was removed from the shader but this made it difficult to differentiate the higher points from the lower points:

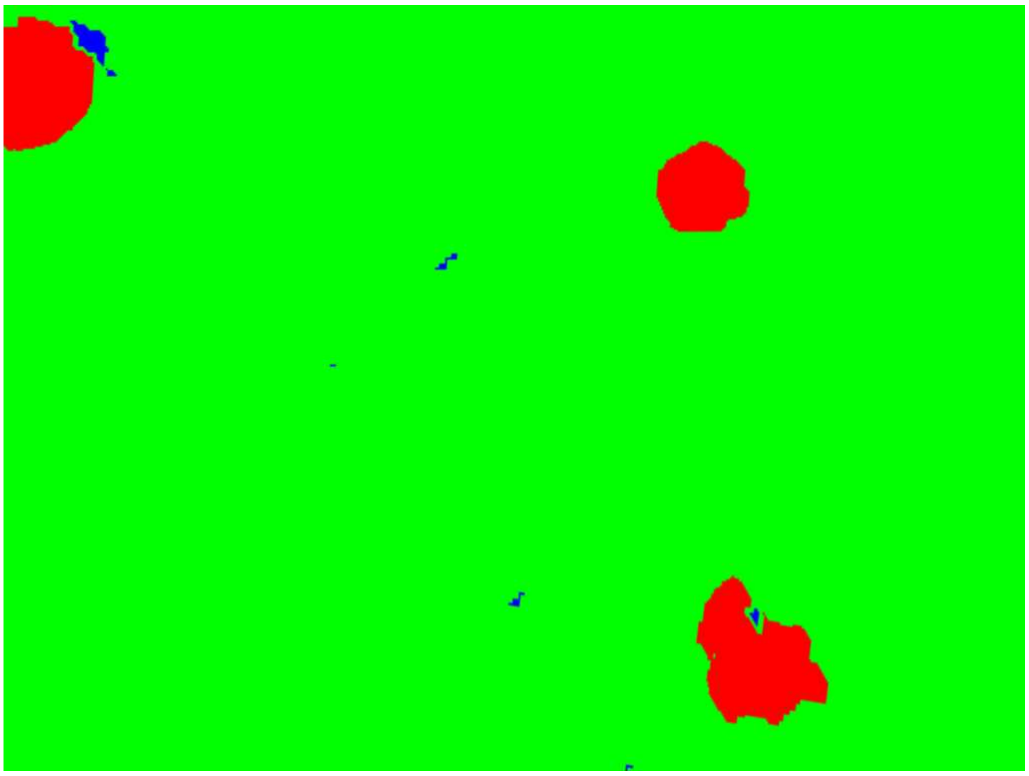


Figure 18: As opposed to Figure 17, it is now very hard to discern depth from the visuals

We then attempted to interpolate between the different levels, which did not make a big difference as the change between two places was too small to notice:

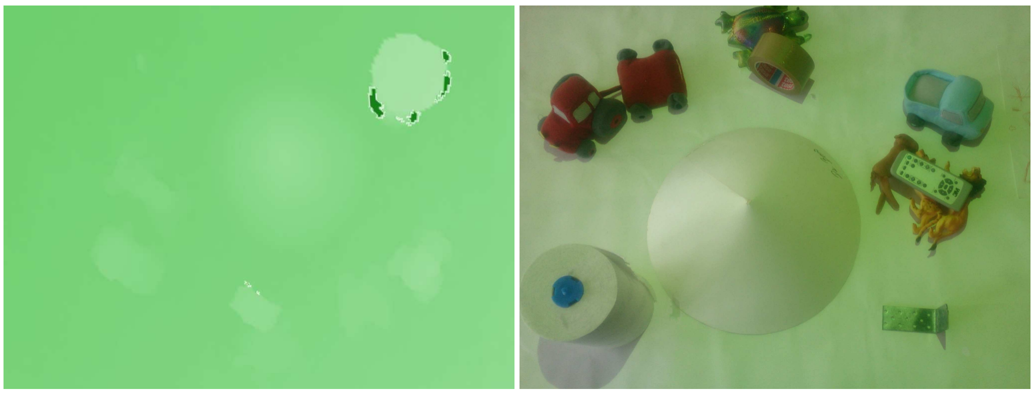


Figure 19: With the linear interpolation shader it is now easier to see contours of objects on a monitor, but it is hard to see any difference on the surface where we project the image (note the cone hat)

Drawing inspiration from the original SARndbox we decided to draw on some simple topography lines in the shader as this would instantly make it visible that there was height differences, even if projecting a heightmap on a flat surface:

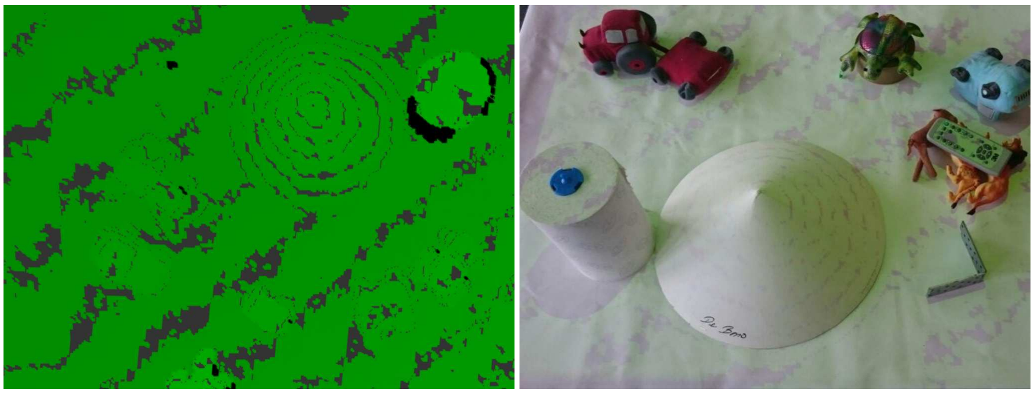


Figure 20: With topography lines, even while changes in color are still small, it is much easier to notice that there is a height differences, and it is also easier to see the effect it has on objects in the sandbox (note that the contour lines)

As can be seen there is still considerable room for improvement in the shader. However, do note that the thick lines going across the plane is, at least partially, due to the kinect camera not being parallel to the surface.

5.1.2 UV Mapping

After the kinect loads the user needs to choose a quadrilateral out of the image shown, where the actual table with sand is. This is done so that the kinect could be slightly angled or misplaced, and the user would still be able to calibrate the screen so it shows only the part of the sand. Clicking with a mouse the user can set out crosses where the middle of the crosses will be the actual point in the code, after the first click a line will be drawn from the previous point to the next point, and on the last (fourth) click the four click will be joined into a quadrilateral. This quadrilateral then needs to be rendered into a rectangle fitting the screen, and we had some challenges doing just that. If the user needs to reset his clicks he can press the "c" button on the keyboard.

First, the Irrlicht documentation for UV mapping was nowhere to be found. Consecutive searches with search engines and Irrlichts own forum-search were not enough to actually figure out how to do it. In the end we decided joining #irrlicht on freenode (IRC) and asking there, and a nice guy named CuteAlien (whose presence has been seen on the forums) helped us out! It seemed we were using one of the few game objects from Irrlicht that does not support UV mapping; a billboard node. We switched over to our own SMesh and things worked fine.

In essence this is what is happening; The data read in from the Kinect makes it easy to see the table, but we want to only project the relevant parts of the data, this is where the calibration comes in:

Figure 21: You can slightly see the outline of the table, with the user clicked crosses and lines.

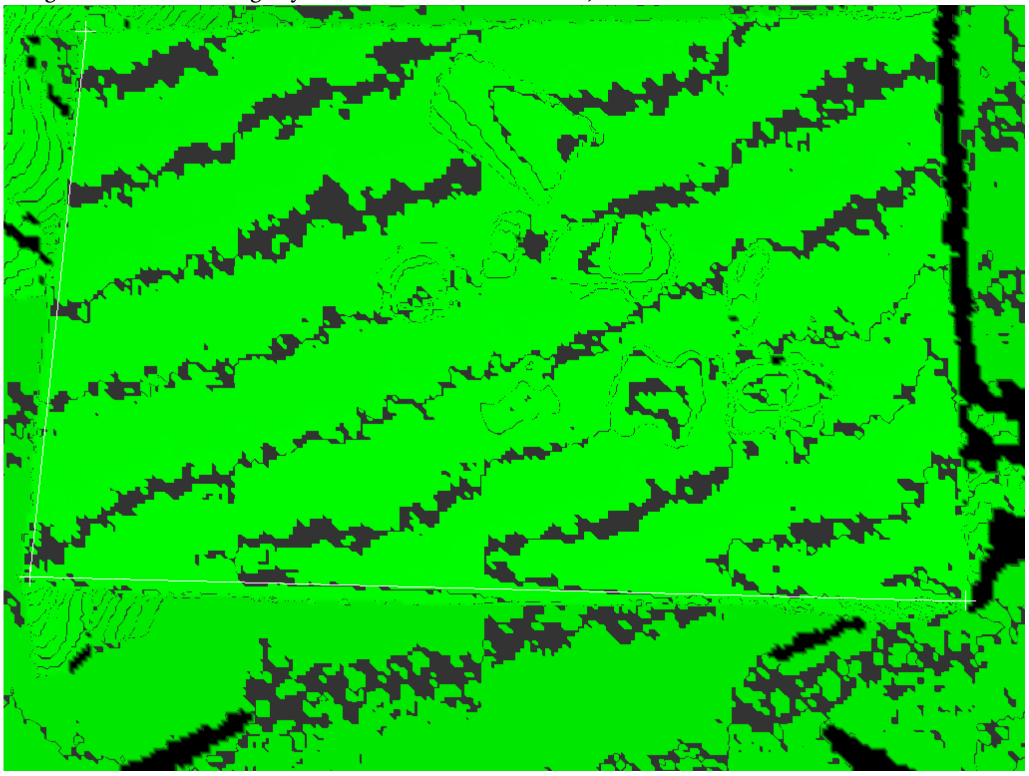


Figure 22: And this is how the result from the UV-mapping looks like.

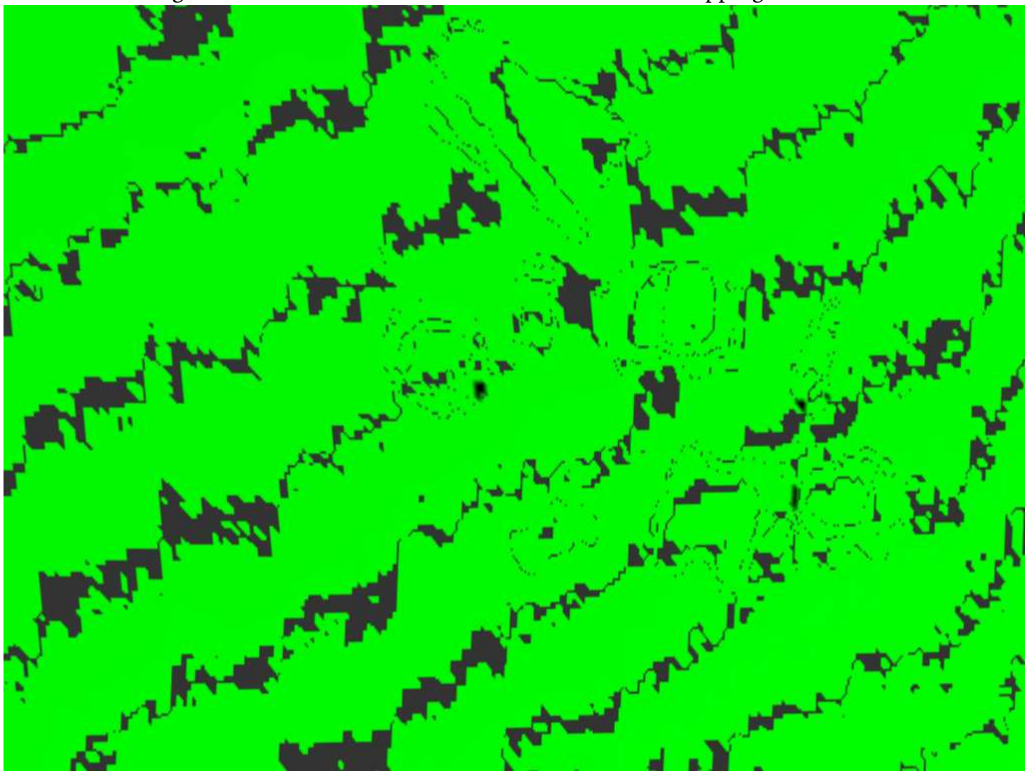
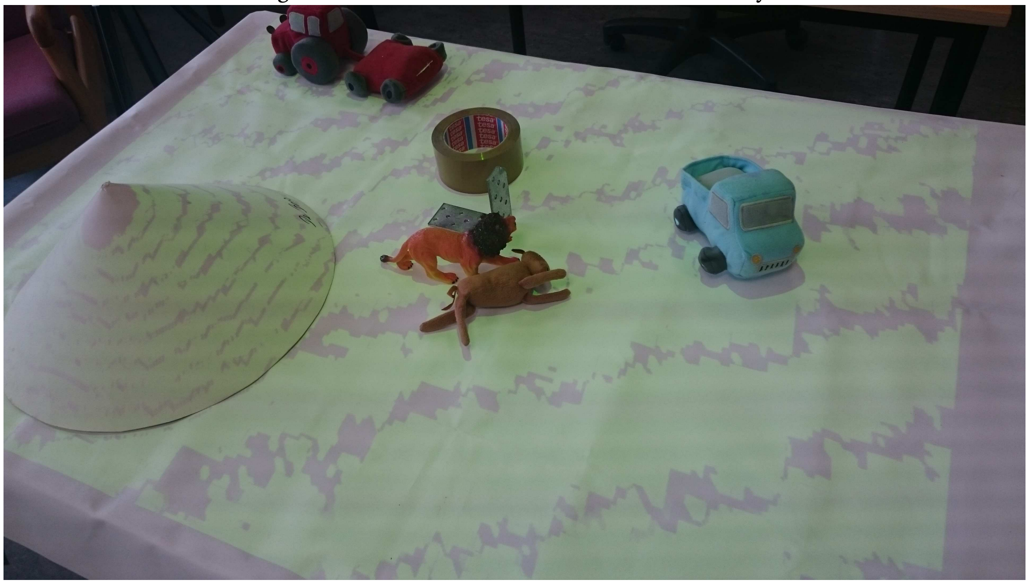


Figure 23: And here is how it looks like in reality.



5.2 Kinect

This section deals with interesting problems and challenges revolving around how we got the kinect device working and how we pull data out of it.

5.2.1 Working in a virtual machine

We initially did not have a dedicated desktop with Linux to work with the Kinect. This meant that Mårten, who was using Windows, had to use a virtual machine to emulate the target environment. While testing, however, the feed from the camera would always be distorted, not at all resembling what the camera should have been seeing. We could not find the specific cause or any solution to this and the issue never appeared on a machine which was directly running a Linux distribution.

5.2.2 Getting the kinect to work

Getting the kinect to work was an interesting journey. At first we tried using Oliver Kreylos' programs, and found that the programs would not find the kinects. After a long time searching we finally found that in one of the GNU/Linux files, a parameter needs to be set so that the kinect will not suspend after a while or immediately after being plugged into the computer. This parameter is in /sys/module/usbcore/parameters/autosuspend and needs to be set to -1. After doing this, both Oliver Kreylos' programs and libfreenect with libusb were working (with some bugs here and there, of course). To fix having to set this up every time a machine was rebooted we added an entry to the machines GNU/Linux crontab that did this for us on reboot, like this:

```
@reboot echo "-1" > /sys/module/usbcore/parameters/autosuspend
```

Figure 24: Kinect Reboot Cronjob for Autosuspend Off.

5.2.3 Reading in terrain data

In the specifications for the kinect, both Mr. Kreylos and Microsoft said that the kinect would read in a greyscale image that was 640x480, which can be used as a heightmap, and that also seemed to be the case when we used the same data in our irrlicht mock-up programs. This then led to total confusion when trying to export the same data into a .bmp (bitmap) image file; the file always had a completely dark stripe with zero-values to the right on the image. After digging for a while, we found a small text on openkinect.org saying

> The frame output of the depth camera is 640x480 (the rightmost 8 columns are always "no data", so you get an effective potential image size of 632x480 in a 640x480 buffer) [15]

This cleared our confusion, but we found no reason as to why it was made like this. In the end we did not do anything to "correct" this behavior as it was easier to deal with the heightmap while it had a 4:3 ratio, which matches the output resolution's ratio.

5.2.4 API

The kinect API is a simple C++ .h class (plus an empty .cpp file because it cannot be compiled without a .cpp file) that inherits from libfreenects FreenectDevice class. There is a constructor, a DepthCallback and a getDepth function. The DepthCallback is called on by libfreenect every time the kinect data refreshes, all we do is copy the data received into a uint16_t vector. The getDepth function is the main and only attraction to the API, it returns a bool but takes a pointer to a uint16_t vector as argument and swaps its own kinect data buffer into the one pointed to, then returns true if it was able to do that. The API has mutexes to prevent segmentation fault, using standard C++14 lock guards which unlocks the mutex when the function goes out of scope.

5.2.5 Kinect Shadows

Since the kinect works by creating infrared dots on everything it can see and then uses these dots to calculate depth by recognizing patterns in the collective dots, sometimes data is either lost or shadowed over by other objects. When this happens there will be a giant hole in the terrain we create, where the data points will be close to zero in value. We never tried fixing these so-called shadows, but we have played with the thought of averaging between points at the edges of the hole in order to patch it up. The first problem with that is that it would skew the actual data significantly. For example, if there is a shadow right next to end of a slope, the shadow would turn into a steeper slope itself. Because of the scope of this problem we decided not to focus on implementing a fix for it. Although a possible solution is to recursively average the height the middle point of where the shadow is located with the closest non-shadow values.

5.3 Networking

This section describes interesting challenges and problems we had dealing with networking while doing the project, along with options for how we could have solved them and how we ultimately went about solving them.

5.3.1 Network Library

For our networking we needed reliability and in-order similar to what TCP provides but without the naivety of TCP itself. We were left with two choices; find a library which gives us what we needs or implement it ourselves on top of UDP. Having previously implemented the concept of 'connections' on top of UDP in an earlier project for the Game Programming course we knew that implementing this would be very time-consuming. So we decided to look at a few different networking libraries, a comparison of these libraries can be found in the table below:

Library	Reason
POCO [16]	Contains many unneeded features, does not contain reliability without using TCP or implementing ourselves
Ice [17]	Could not find anyone who recommended this, which lead us to not choose it
RakNet [18]	More features than needed (e.g. lobby system, object replication, voice communication)
Asio [19]	Would have to use TCP or add the required features ourselves
ENet [20]	Thin layer on UDP which grants reliability and in-order packets

Table 1: Evaluation of network libraries.

While we used ENet for our networking initially we later got a requirement to have an API which was easy to interface with from any language. Because ENet is a custom protocol on top of UDP which primarily only supports C and C++ with some unofficial wrappers available for Python [21], Godot [22], Java [23] and C# [24], we came to the conclusion that ENet was too restricting to use for this requirement. Because of this we ended up implementing a TCP server using the Asio library.

5.3.2 Delta Change

The full heightmap is about 1 megabyte in size ($640 \times 480 \times 4\text{bytes} = 1228800\text{bytes}$). This is a considerable amount of information to send and process regularly. Because of this we defined a system where we would only send the indices which changed and the difference in value.

```
message DeltaChange {
  required uint32 index = 1;
  required float difference = 2;
}

message TerrainUpdate {
  repeated DeltaChange deltaChange = 1;
}
```

Figure 25: The messages we used to support the delta change system

This worked nicely in testing with fake data, but we failed to foresee the amount of noise which the kinect sees. The end result was that delta changes used almost twice as much data on every transmission (since there was now a 4-byte index to send along with each 4-byte value.) Upon discovering this we disabled the delta change system and opted to always send the entire terrain. This testing was done *before* we implemented averaging in our Kinect library and we have not had the time to make it work again to see if it improved.

5.3.3 Unknown Packet Content

With protobuf we had to parse the protobuf message from an array to start using it. This proved to be a problem when you do not know which message the packet contains. A possible solution was to send a flag before the protobuf message to let the other party know which type of message would follow, but did not use this in the end because there was a better solution proposed: A union message[25]. Because protocol buffers has the ability to put messages as a field in other messages we could then create a message which contained all other messages along with a flag to signify which message it contains. The messages it could contain would have to be marked as 'optional', so they could be properly serialized without containing these messages. Below you can see our union message for the project.

```
message Union {
  required uint32 flag = 1;
  required uint32 typeFlag = 2;
  optional FullHeightmap fullHeightmap = 3;
  optional TerrainUpdate terrainUpdate = 4;
}
```

Figure 26: Our Union message, which includes other self-defined messages (FullHeightmap and TerrainUpdate.)

5.3.4 Unknown Size of a JSON Message

To properly parse a JSON-encoded message we had to ensure that we had the entire message or the parser would fail. Because TCP is not an object-oriented stream[1], we did not natively have any way to see if the entire message had been received or not. During preliminary research we found a couple of ways to solve the problem. One way to solve it was to have unique delimiters in front and after each message, and another way was to simply include the length of the JSON message in front of the message. We opted to go with the latter as it was easy to get the message's length, and it is easier than selecting a delimiter which then cannot ever appear at any point inside the message. After reading in the message length, which is delimited by a carry-return and new line (\r\n), we then continue reading until we have read enough characters to match the message length or until the stream ends (the other party disconnected.) We can then confidently parse the message and retrieve the content.

5.4 Game Engine

We needed a game engine, so in order to determine whether or not a game engine was suitable we started looking for documentation, comments, and forum posts that would back up these points:

[1]An object-oriented stream would only return complete objects and not partial objects

1. Ability to change terrain while game is running
2. Lightweight
3. Cross platform; At least Windows and GNU/Linux
4. Easy to set up; packages for the platforms above, optionally easy to compile
5. Easy to get started and expand
6. Can use C++ with it, not a scripting language

We were recommend to look at an open source game engine called Torque 3D, but neither of us were able to compile it on our machines. Here is a table of engines we investigated with evaluations for each:

Engine	Reason
Unity	Initial research showed no real-time changeable terrain, C#
Unreal	No real-time changeable terrain
OGRE3D	No real-time changeable terrain,
HORDE3D	No real-time changeable terrain
Panda3D	For GNU/Linux it only works on debian based distributions
Irrlicht	Lightweight, real-time changeable terrain, C++, cross platform
Godot Engine	More of a scripting based engine
Torque3D	Cannot compile
Xenko	C#
Maratis	Cannot compile, forum says static terrain/triangles
Delta3D	"Deformable terrain would be difficult to implement"
Shiva	Only Windows and Mac commercial

Table 2: Evaluation of game engines.

5.4.1 Irrlicht

Based on the comparative evaluation of game engines, Irrlicht was selected. We were able to set up a small test on changing terrain while the game was running, which worked. What we like about it is that it is lightweight, we can program in C++ with it, easy to get started – many tutorials, mature forums, packages in most common GNU/Linux distributions.

5.5 VR Game

After months without VR hardware, we came to the point where we essentially had to hack together some sort of VR. So we looked at our options and tried finding one that was simple and fast. We could have made our own, but this would have taken a lot of time; we would have needed to create drivers for the VR hardware, the Oculus Rift, on GNU/Linux as well. We chose not to do that and instead look for engines that already had integrated VR. As said, the most natural path to take from here was to use an already VR-enabled game engine; Irrlicht does not have any built-in VR but there was a project that was made in Irrlicht using VR with Oculus' SDK – it was on Windows only however so we dropped that one, as both of us needed to work on the VR project. Therefore, we looked at Unity again and the documentation said in order to enable VR in Unity you just set a checkbox and that should have been it:

> Enabling Unity VR Support
>
> To enable VR for your game builds and the editor, set the "Virtual Reality Supported" option in Player Settings. [26]

The decision was then made to use Unity, as it already had VR ability, and just build on top of an already existing project. We found CarSimUnity[6] and asked for permission [Appendix I] to use it. We then proceeded to add an editor script called HeightMapFromTexture [27], this way we could add a heightmap image gotten from our VRterra server using the kinect and import it into the unity project as a texture, run the editor script, and it would put the texture in as a terrain into the scene. We also edited the car and moved the camera into the interior of the car. It all seemed to work, and we thought we only had to do what the Unity documentation told us to do to enable VR with it.

This was not the case. According to Johannes you cannot just "enable VR" in unity, you need to set it up. Luckily he helped us set up a new Unity project using VR with a moveable character, enabled for the Oculus. We then proceeded to add the HeightMapFromTexture script again and test walking around in a loaded terrain, here is a screenshot of that:

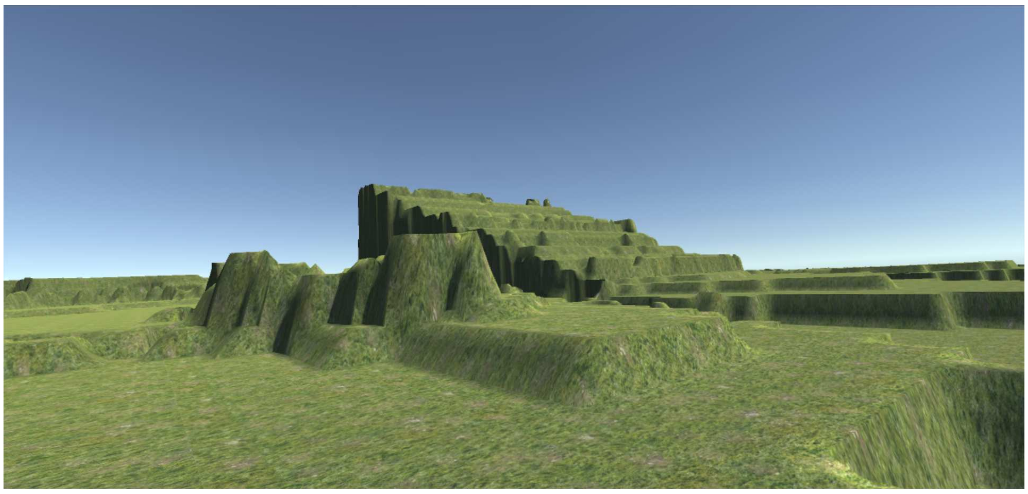


Figure 27: Screenshot of how terrain looks like in the VR Demo.

A review of how walking in the terrain was can be found in section 7.2

6 Deployment

6.1 Deployment Alternatives

In this section we will talk about ways to deploy, in section 6.2 we talk about how we chose to deploy.

6.1.1 Docker

Docker is an open-source project that automates the deployment of applications. You can build, ship, and run any app, anywhere. Docker uses a technique commonly referred to as containerization, this means Docker will start a new "container" in which the things you are to deploy will be stored and run – think of a virtual machine, but without the virtualization. The container will contain, or jail, the software and run it in its own container environment. It produces a more secure (but not completely secure!) way of testing and deploying. A dockerfile will specify what to include into the container, and a dockerfile can contain scripts to do a bunch of things; install from package manager, download things, compile things, etc.

On Windows, Docker instances are actually running in virtual machines running GNU/Linux with Docker in them although this may change after the recent news about Windows Subsystem for Linux in Windows 10.

With Docker we could then create a script to boot a supported base image in a container and then automatically fetch the source code of the latest stable release, the required libraries and then build and launch the application when the container starts up. In addition, the program will always run as intended, no matter which environment the container is running in. The only downside is that it is not user-friendly to connect USB-devices (the kinect), the display for visual output or persistent storage.

6.1.2 Packages

On GNU/Linux most programs are distributed using secure package repositories. These repositories usually include a package file that has the binary for the given software and also describes things like where to install, what dependencies to also install, etc. These are ".deb" files on Debian derived distributions and ".rpm" for Redhat based distributions. Essentially they are .zip files with a lot of additional information. There is no need for a repository to distribute package files, you can distribute a .deb file yourself and people will be able to install it and the dependencies.

This solution makes it easy to install our program on the platform(s) which support the package we create. And given that we are targeting a single platform this choice seems suitable. The downside is that it would not work for all GNU/Linux platforms; but in this case it does not actually need to as Mr. Kreylos recommends using Linux Mint – which is what we believe most setups of SARndbox is using (or , at least, a close derivative compatible with .deb).

6.1.3 Automatic Installation Script

We could also create a shell script that will download the source code, compile it into a binary and put it in the users path. Although this would require the user to temporarily install a compiler and download headers before compiling. This should not be foreign to your average Linux users, but it is not an ideal solution for distributing an application. And if support is planned for multiple distributions we would also have to make multiple scripts to take into account the various package managers.

The downside here is it would, again, be very distribution specific (not to mention platform-specific). In addition it would take a lot of time and testing to make sure it works every time. It does however come with the upside of being user friendly, a mere "./configure" and "make install".

6.1.4 Distributing a pre-configured system

Create an image of the machine we set up and distribute that same image, optionally create it as a virtual machine. Although this is an inconvenient method, where the image has to be applied directly to the hard drive, it is still a possibility and would guarantee that the user has the correct setup. If we were to create a virtual machine image it might lead to the issue described earlier in section 5.2.1.

It would be big, might not work out of the box for different hardware, and the user would need a hard-disk available to write over. Probably not that user-friendly either.

6.2 Implementation

In this section we talk about how we chose to deploy.

6.2.1 Packages with CPack

Since the requirement Mr. Kreylos set for his SARNDbox setup was GNU/Linux Mint – and all other setups with his software is using that as well – we figured that creating a .deb package one could install on their installation would work well enough. Interesting enough, CMake has a packaging system integrated within itself called CPack. With CPack you can create .zip, .deb, .rpm and many more fileformats (even NSIS for Windows and Mac OS X packages). It took a day to figure out, but we eventually got it up and working – meaning we were able to create installable .deb packages. To install the .deb package you just run these commands in your shell:

```
$ dpkg -i vrterra.deb
$ apt-get install -f
```

Figure 28: Commands for installing vrterra.deb

The first command will install vrterra and the second command will install the dependencies you need in order to run it. Optionally one could use the gdebi tool:

```
$ sudo gdebi vrterra.deb
```

Figure 29: Command for installing vrterra.deb with gdebi

Compared to the other deployment alternatives we found this to be the easiest to do, yet with a certain insurance that it would work for the platforms we targeted, even though it is distribution specific.

7 Tests and VR Testing

7.1 Unit Tests

In this section we will go through the different types of tests we wrote, split into what the tests were for.

7.1.1 Kinect

TestDepthBuffer

The kinect has a simple test that will try to connect to the freenect device driver using our VRterraFreenectDevice class, it will then try to start up the kinect and read some data from it – if any of the data returned is negative, or if the device doesn't start up, the test will fail.

7.1.2 NetworkUtils

TestComparePeers

TestComparePeers tests NetUtils' ComparePeers to assert that ENet peers are the same by creating four peers, two alike and two different, then trying to compare them. If any of the peers are not what the comparison expected then the test will fail.

TestApplyFullUpdate

TestApplyFullUpdate will create a small terrain, resize the heightmap, run applyFullUpdate from NetworkUtils and assert that the values on each index is what they should be (incrementing chronologically from 0 to 3). If any of the values are not what they should be the test will fail.

TestExtractProtobufUnion

TestExtractProtobufUnion test that extracting the union from an ENet packet works, checks the flag and typeflag of the packet.

7.1.3 Terrain

TestResizeTerrain

TestResizeTerrain does just that, tests if resizing the terrain works by using setNewSize and asserting that the size, width, and height of terrain has changed. If either one of size, width or height are not what they supposed to be the test will fail, otherwise it will succeed.

7.1.4 Config

TestGetValue

TestGetValue defines a JSON string and uses Config from Utils to parse this string. It then parses this string and asserts that all the values tested are the same as in the JSON string as well as attempting to load a few values which do not occur in the string to test getValue's ability to return a self-defined or standard default value. If any of the asserts fail the test fails.

7.1.5 Container

TestRemove

TestRemove initializes a vector of integers with 1's and 2's, and also a second result expectation vector of integers with 2's. Same amount of 2's in both vectors. It then tries to remove all 1's from the first vector and asserts that the size is the same as the second expectation vector. Afterwards it checks if the content of both vectors are the same, and lastly it removes the remaining 2's from the vector and checks that the size is zero. The test will fail if the assertions fail.

7.2 VR Testing

The VR aspect of the project was tested by creating a small VR game, with a terrain and a player, that was capable of using the oculus rift for VR. We added the HeightmapFromTexture-script [27] and loaded the following image into the project:

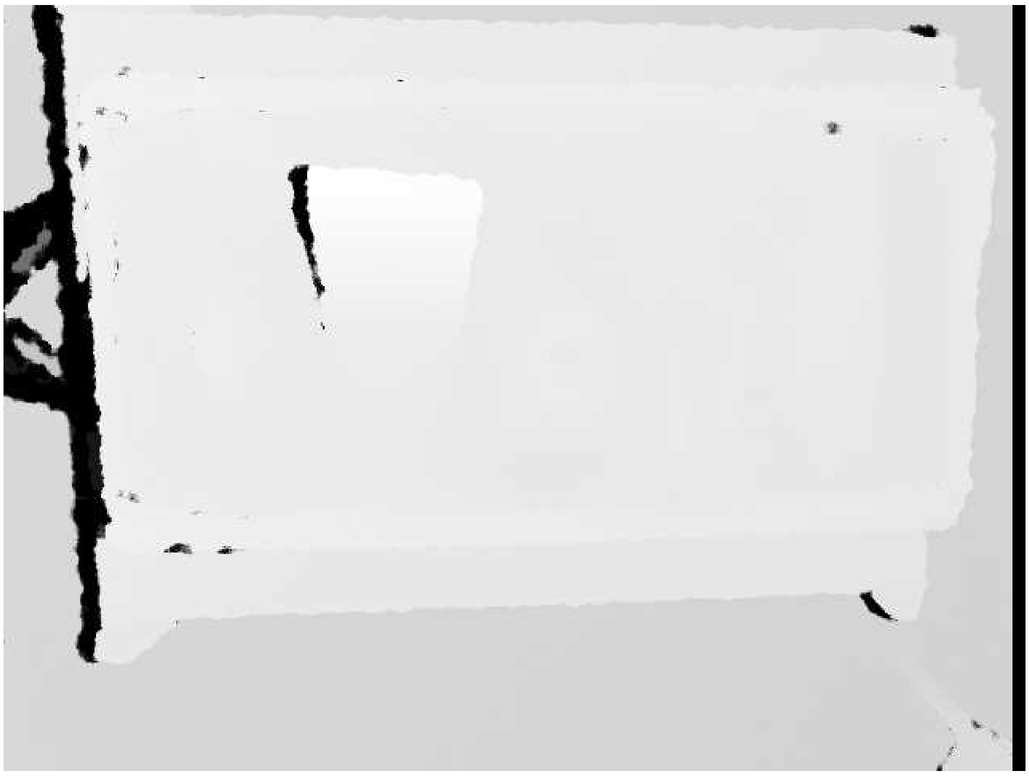


Figure 30: Heightmap loaded into the VR Demo.

Figure 27 is a picture of how this looked like in-game.

We found walking around and looking on the terrain fun, it was interesting to see the terrain in virtual reality; one would recognize certain "landmarks" (such as the giant slope in figure 27). The base movement was very slow so we tried turning this up, something that resulted in dizziness if someone else were controlling the movement.

Here is a picture of how it looked like through the lens of the oculus rift:

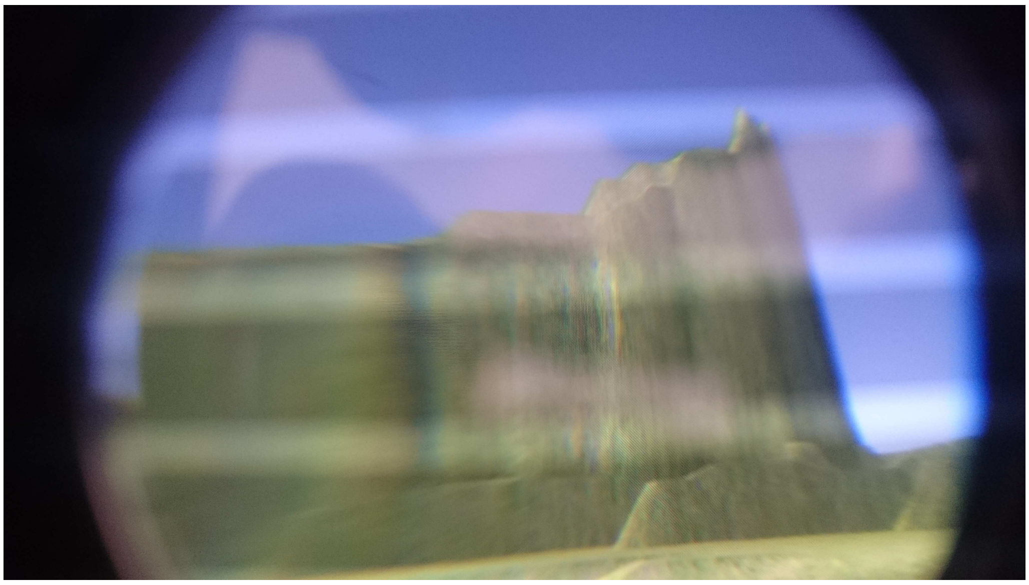


Figure 31: Image of VR Demo through Oculus' Lens.

8 Discussion

8.1 Results

8.1.1 Engine

Choosing an engine that was sure to work was challenging. As mentioned, we started creating our own engine in OpenGL/SDL2, but the initial phase of development took longer than anticipated and so to create the full engine went beyond the scope of the project. After evaluating different engines, as seen in table 2, and choosing Irrlicht, we could start focusing on features.

There were some minor confusions with Irrlicht, we could not find a function for updating normals but by using a setScale function with the same scale (getScale) as parameter the normals would be updated, and on top of this the mesh itself would not reflect the changes made to it unless the terrain was moved. This forced us to perform a virtual move where we moved it to its current location. This was not as intuitive as it should have been. There were other similar confusions, like with the UV mapping we talked about in calibration implementation and updating collisions.

While the bad sides are easy to list since they made a larger impression it was also a nice engine to work with. Setting up a window, running the game loop and drawing was very easy and straight-forward.

8.1.2 The Editor

We have mentioned the editor both at the start of this paper and also in the project plan appendix. We stopped working on the editor after a couple of months, when we got the kinects up and working. It was initially made as an alternative to the Kinect and it let you alter the terrain by clicking on it with your mouse. After getting a kinect there was no good reason to keep working on it so we converted it into what is not called "NoKinect", which is the same application as the projector application except that it generates fake data by altering the heightmap at random indices. It is now merely a way to test the shaders and to see if the libraries which Kinect and NoKinect has in common still compiles and works properly. Looking back at it this application (Editor and then NoKinect) was always an invaluable asset to the development, letting us simulate terrain changes in different manners and seeing how our system handles these changes. Early in the editor revealed that the normals do not get updated in Irrlicht's terrain unless it is rescaled and without it we would not have known this until we got the Kinect.

8.1.3 Projection

In this section we will discuss the projection application in VRterra.

Calibration

To calibrate VRterra the user needs to click four points on the screen, simulating the edges of the projection onto the sandbox. When the user clicks the last point VRterra projector program will do a UV mapping on those four points; scaling and transforming the quadrilateral into a rectangle the size of the projector 4:3 screen. This of course means the sandbox needs to be static and not move, but that should not be a problem (it is a sandbox, after all). When the sandbox doesn't fit the video projection surface, being too big or too small, it is difficult for the user to calibrate it. Although UV mapping helps considerably, making it fit *exactly* was very difficult while we were testing it. Because of this the projection is often skewed a little. If the kinect is angled it will affect the data read in as well, the sand furthest away (with most angle) would be either be at a lower or higher point than the sand closest – effectively tilting the whole table with the angle of the kinect.

Shader

There are several differences in shaders, you can see an example of the SARndBox shader in figure 16, and our early shader in figure 17. While we are satisfied with the result we got out of it in the end, as can be seen in figure 20, we do realize that there is a lot of room for improvement, notably the choppy topographic lines. Along with this the color changes should also be more noticeable between the lines in the topography.

Setup

The sand we had during our final test setup was too dark, making it difficult to see the projected image on the sand. In addition, the lighting in the room itself was too bright and the room did not have any curtains to let us block out the light.

8.2 Group Work and Workload

As is usual in projects we split the work between us, so each of us ended up becoming more familiar with certain areas of the code than the other. Because of this all issues regarding a piece of code in one of these areas had to be handled by the person who wrote it. This was especially true of the network code. During the work we had some disagreements on how things should be solved, and due to neither of us taking on the leader role we always solved these disagreements by convincing the other person that their own solution was better. Since it usually was.

We would both work from 8 to 15 every weekday. This meant that we would get 7 hours per day which was the amount of hours needed to reach the expected hours of work. Some days were slow and rather unproductive compared to other days that seemed to end just as they started with much work done.

8.3 Further Development

8.3.1 Custom marker texture

Current the marker only displays as a red circle. Adding the ability to supply a custom texture would require work in the TCP server to support receiving an image as well moving this image into an instance of Marker and then loading the image into Irrlicht before drawing it.

8.3.2 Marker cleanup

The markers currently do not go away. This means that if an application which creates a marker needs to restart there will be a marker on-screen permanently until the sandbox has been rebooted. This could be fixed by having markers disappear after not being updated after some specific amount of time. This would also require implementing a response to trying to update a non-existing marker to notify the client that the marker does not exist.

8.3.3 Exporting as map to Minecraft

There already exists programs that convert heightmaps to Minecraft maps, so with that in mind, it would be really easy to set up a small program that gets the heightmap in our server program via TCP and then uses one of these heightmap to minecraft map programs to convert it.

8.3.4 As an API

One of the requirements for the project was that we needed to offer other developers with an API. That way one could start running the server with the projector, kinect, and sand, and create a program that pulls the terrain data out of the server. There are two parts to this; Kinect.h and the TCP library. With Kinect.h (as long as you fill the dependencies) one could include the kinect stuff into their own C++ projects, further using the data returned by the Kinect for anything they want. With the TCP library you could potentially use any language that supports JSON and pull data from the server. This means you could use any game engine and any language – provided you have a server.

This is great because there is a lot you can do with this; create games and applications that go beyond the scope of this project. Maybe you want to be able to 3D print terrain gotten from the Kinect, or maybe you want to create a game with custom sandbox-made levels. Another example is our idea that was discussed with the Hamar Game Collective in section 2.2.4, or any of the other ideas in the sections above that.

9 Conclusion

When we started this project we went into it wanting to do something experimental and creative, thinking it would not overscope too much so we could focus on the good features.

When we were looking for an engine, before we started developing, we came up empty-handed. It turns out we would have found Irrlicht with a little more extensive research, which would have saved us the time we used writing the OpenGL code in the beginning.

The kinect framework seemed simple enough; read in the image from a kinect and edit a mesh to match it. The task being this simple would have let us focus our effort on the game idea, however it was not as easy as we thought it would be. When we realized how long it would take we restructured the project and the major goal of it into mostly focusing on the sandbox and the API.

The VR functionality in Unity, originally thought to be a mere checkbox to enable/disable VR, proved to be more complicated than originally thought. It showed us that sometimes even documentation can be wrong and that the only way to know for sure is to test it.

There were times where we were stuck in a rut and had a hard time starting to work, which undoubtedly wasted time, but we always pulled through in the end and got the work done. Some days we did not feel like working at all but we took it like a job and in jobs you do not get that luxury.

Not only did we improve our coding and team work experience, we also think it was a helpful experience to work with an engine that was unknown to both of us, and also using multiple libraries we had never used before.

Bibliography

[1] Kreylos, O. 2016. Augmented reality sandbox. http://idav.ucdavis.edu/~okreylos/ResDev/SARndbox/. Accessed: 2016-04-20.

[2] SMARTMania.cz. 2011. Interactive kinect sandbox. https://www.youtube.com/watch?v=8p7YVqyudiE. Accessed: 2016-04-20.

[3] Monobanda. 2011. Project mimicry. http://mimicry.monobanda.nl/. Accessed: 2016-04-20.

[4] Kreylos, O. 2016. Sarndbox instructions. http://idav.ucdavis.edu/~okreylos/ResDev/SARndbox/Instructions.html. Accessed: 2016-04-22.

[5] Google. Protocol buffers | google developers. https://developers.google.com/protocol-buffers/. Accessed: 2016-05-13.

[6] Szewczyk, R. Carsimunity: Simple unity car simulation. https://github.com/radx64/CarSimUnity. Accessed: 2016-05-04.

[7] Brown, D. math - treating a 1d data structure as 2d grid - programmers stack exchange. http://programmers.stackexchange.com/a/212813/211564. Accessed: 2016-05-12.

[8] Taiga.io. Taiga.io. https://taiga.io/. Accessed: 2016-04-20.

[9] Wikipedia-contributors. 2016. Agile software development - wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Agile_software_development&oldid=715998631. Accessed: 2016-04-19.

[10] Atlassian. Bitbucket - the git solution for professional teams. https://bitbucket.org/. Accessed: 2016-05-18.

[11] Toggl. Toggl - time tracker & employee timesheet software. https://www.toggl.com/. Accessed: 2016-05-11.

[12] Irrlicht-Team. Irrlicht engine - a free open source 3d engine. http://irrlicht.sourceforge.net/. Accessed: 2016-05-11.

[13] Irrlicht-Team. License - irrlicht engine - a free open source 3d engine. http://irrlicht.sourceforge.net/?page_id=294. Accessed: 2016-05-11.

[14] Kitware. Cmake. https://cmake.org/. Accessed: 2016-05-11.

[15] OpenKinect-Contributors. Faq - openkinect. https://openkinect.org/w/index.php?title=FAQ&oldid=1908. Accessed: 2016-05-12.

[16] GmbH, A. I. S. E. Overview | poco c++ libraries. http://pocoproject.org/. Accessed: 2016-05-12.

[17] ZeroC, I. Ice - comprehensive rpc framework. https://zeroc.com/products/ice. Accessed: 2016-05-12.

[18] LLC, J. S. Raknet - multiplayer game network engine. http://www.jenkinssoftware.com/features.html. Accessed: 2016-05-12.

[19] Asio. asio c++ library. http://think-async.com/. Accessed: 2016-05-12.

[20] ENet. Enet: Enet. http://enet.bespin.org/. Accessed: 2016-05-12.

[21] Resch, A. aresch/pyenet: A python wrapper for the enet library. https://github.com/aresch/pyenet. Accessed: 2016-05-12.

[22] jrimclean. jrimclean/gdnet: An enet wrapper for godot. https://github.com/jrimclean/gdnet. Accessed: 2016-05-12.

[23] Marshall, C. csm/java-enet-wrapper: Java/jni wrapper around enet. https://github.com/csm/java-enet-wrapper. Accessed: 2016-05-12.

[24] Shoffner, N. Nateshoffner/enetsharp: C# enet wrapper. https://github.com/NateShoffner/ENetSharp. Accessed: 2016-05-12.

[25] Google. Techniques | protocol buffers | google developers. https://developers.google.com/protocol-buffers/docs/techniques#union. Accessed: 2016-05-13.

[26] Unity-Technologies. Unity - manual: Vr overview. https://docs.unity3d.com/Manual/VROverview.html. Accessed: 2016-05-11.

[27] Haines, E. Heightmapfromtexture - unify community wiki. http://wiki.unity3d.com/index.php?title=HeightmapFromTexture&oldid=13516. Accessed: 2016-05-11.

A Project Plan

VRterra

Mårten Nordheim & Nichlas Severinsen

28.01.2016

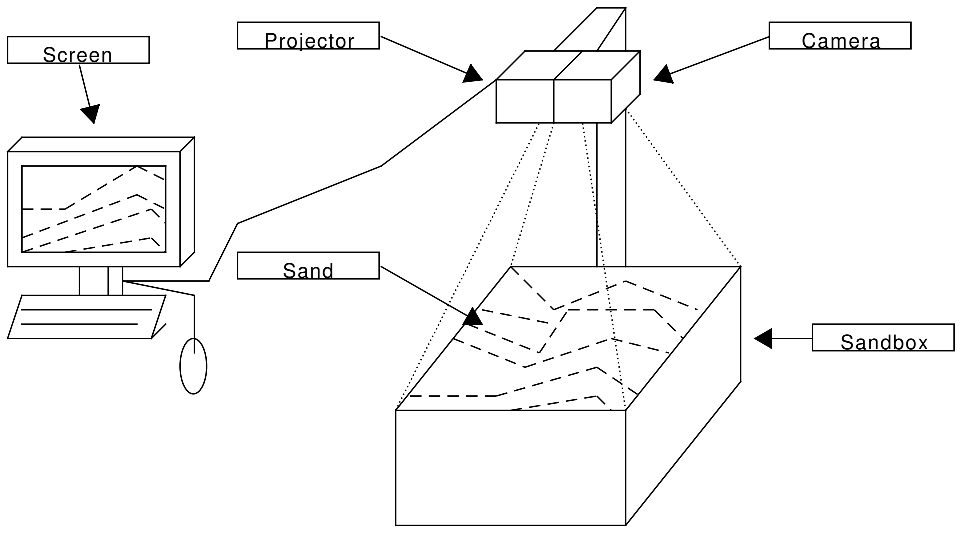
A.1 Goal

The goal of the project is to create a virtual reality game with real time terrain input. The input will, in the beginning, come from a tool we create. This tool will later be replaced by a sandbox with a projector and camera. Therefore, we have divided the project into multiple parts:

1. Shared libraries for Terrain & Networking
2. Basic Engine
3. Terrain Editor
4. Virtual Reality Game
5. Sandbox setup with camera and projector.

Parts 3, 4, and 5 all depend on part 1 and 2; the terrain library, the networking library and the Engine.

A.1.1 Visualization



A.2 Background

The augmented reality sandbox has been built and used by other people earlier, but few have made a game based on it. As we thought it could be interesting to have a game based on the sandbox we decided to do this as our project.

A.3 Task Description

The project task is to create a virtual reality game, possibly with co-op, that can take real time terrain input. This way one player can control the environment and terrain of the other player, it is all based on a heightmap where too low means water, in between means grass, higher means stone/mountain and at the top there would be snow. All programs will be coded in C++14 and since we have two different operating systems (Windows and GNU/Linux) we will be using CMake as our build system.

Because we currently do not have a sandbox, camera, or projector, we will be making a terrain editor first, which would pipe the terrain into the game via networking. If done this way we can also easily switch out the terrain editor whenever we get a sandbox, by making the sandbox module that grabs terrain data with the camera from the sandbox, projects the terrain back onto the sand with the projector and finally sends the terrain data to the game.

The game itself needs to have recognizable elements of gameplay, most of which will be described in a separate game design document.

A.4 Roles & Responsibilities

- Mårten Nordheim
  Programmer, team leader
- Nichlas Severinsen
  Programmer, designer

A.4.1 Rules & Routines

We will meet with our supervisor, Simon McCallum, every Thursday(?), unless impossible for either party. Internally in the group we will have a daily stand-up. We work between 08 and 15, if one of the group members have to take a day off for some reason then they must notify the other member in a reasonable amount of time before the usual meet-up time. Code should not ever be directly committed to master (the only exception is during the set-up phase.), see git branching for how we commit. Code has to be properly commented, although if a comment can be removed by altering the code into a more readable version then this is preferable. The only exception is that all classes and all functions must be commented. All code must follow the coding conventions outlined in the code convention document.

Every day in a sprint there will be a daily scrum meeting where we talk about the usual scrum things:

1. What we did yesterday.
2. What was delaying progress.
3. What we will do today.

Every end of a sprint, usually Friday, we will be making a sprint log entry summarizing the sprint.

A.5 Development Process

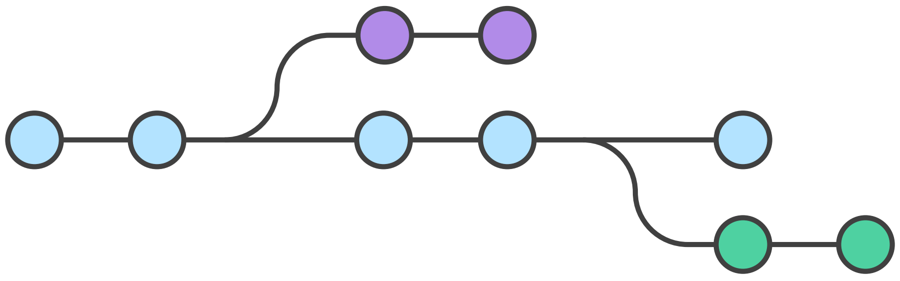
All software will be made with C++14 and there will be cross platform compatibility for Windows and GNU/Linux, where applicable. Virtual Reality may not be enable for GNU/Linux as Oculus dropped support for both OS X and GNU/Linux in May, 2015. We decided on using scrum based on the need for a flexibility that waterfall does not provide, this way there is structure, a plan, and actual tasks that are admitted to each participant. Everything is organized.

A.5.1 Tools

The tools we will be, and have been, using are as follows:

1. Taiga.io: Project management platform for scrum with user stories and tasks/issues.
2. Toggl: Time tracking
3. Bitbucket with Git: Repository, version control system
4. CMake: Cross platform build system
5. Google Test: Testing framework
6. OpenGL, SDL, Oculus Rift VR Framework/OpenHMD, GSL, Google Protocol Buffers (network protocol), ENet (networking), ODE (physics). These are libraries, and there will probably be changes in which one of them we use.
7. Planner: Gantt charts
8. UMLet: UML diagrams
9. Visual Studio 2015: Integrated development environment
10. CLion: Integrated development environment

A.5.2 Git branching



Both of us are experienced with git, so we will be using a workflow called the "Feature Branch Workflow" where there is one master branch, one development branch and whenever someone is working on a feature they will branch off into a different branch named like this: "feature/name". The feature will be merged in using a pull request once the feature is completed. A pull request can be set up before the feature is completed, however, to ask for feedback or assistance on a subject relating to the feature while it is being worked on.

A more in-depth explanation from Atlassian can be found *here*.

A.6 Documentation & QA

A.6.1 Code

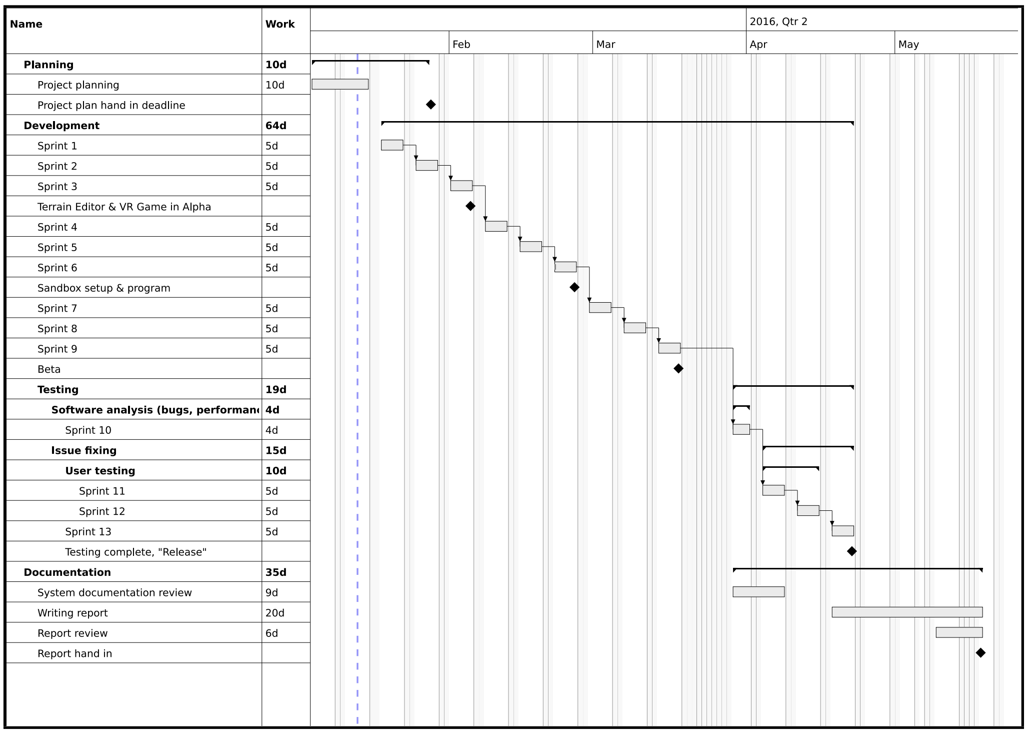
We are writing C++14 code and we have chosen the Javadoc style for source-code documentation, this way we or others can use Doxygen to generate a set of documentation pages in a few different formats, including latex and html. We also have a code convention document.

A.6.2 Report

Documentation will be written during the project, but in the last period of time before the hand in we will solely focus on the documentation and the bachelor report. For quality assurance on the report we will do a review before the hand in, ensuring everything is perfect.

A.7 Development Schedule

A.7.1 Gantt chart



A.7.2 Comments on Gantt chart

Our development schedule is built up by four major phases. Phase one is planning; in the days before the project plan hand in we will be creating necessary documents to help with the development including the project plan and a game design document. Phase two is the development phase split into 13 sprints with four milestones, and inside of that we put the test phase as we will be making fixes and changes. The last phase is our documentation phase where we will focus on finishing the bachelor report.

A normal day starts at 08:00 and ends around 15:00.

A.8 Milestones (Diamond in the Gantt chart):

Milestone 1

Project plan hand in. Basic engine and environment set up.

Milestone 2

Terrain Editor and VR Game in Alpha, meaning "usable, but buggy".

Milestone 3

Sandbox set up and the program for it in Alpha.

Milestone 4

All software in Beta; "working, with some bugs".

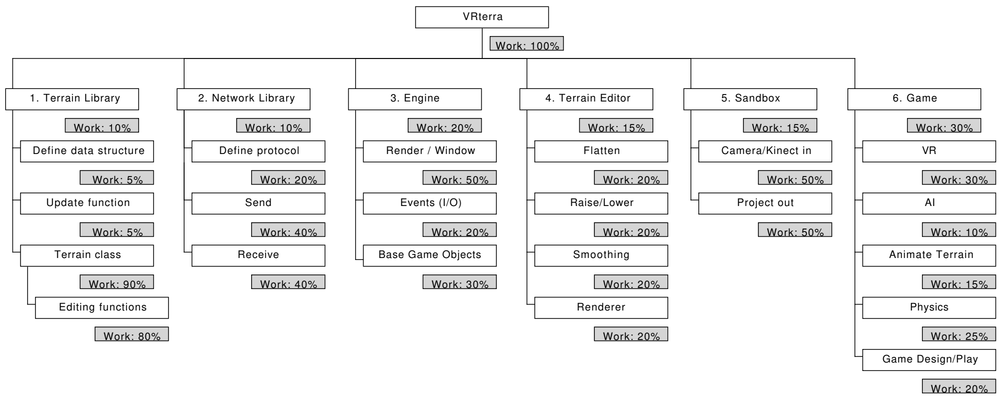
Milestone 5

User testing and issue fixing reaches an acceptable level in order for a "release" to be shipped with automatic installation.

Milestone 6

Documentation and Report done and handed in.

A.8.1 Work Breakdown Structure



A.8.2 Comments on WBS diagram

This is only a draft with estimated numbers and is prone to change as the project evolves. The WBS is structured in layers, meaning the whole project (layer 0) is set to be 100% of the work. Layer 1, the libraries and basic components/parts of the project, together make up 100%, then the other layers together make up 100% of the layer above.

A.9 Risk Analysis

A.9.1 Possible Risks

Name of risk	Probability	Impact
1. Not getting a sandbox set-up	Low	High
2. No VR on GNU/Linux or OS X	High	Medium
3. Not getting VR hardware	Low	High
4. Work or code being lost	Low	High
5. Temporary inaccessibility to Bitbucket	Low	Low
6. Hardware problems, personal or related to project	Low	High
7. Awful look and feel	Medium	High
8. VR/Cybersickness	Medium	High
9. Too much focus on specific parts of the project	Low	Medium

A.9.2 Comments on Risks

1. **Not getting a sandbox set-up**
   There's a pretty low chance we will not get a sandbox set up, in this case we still have the terrain editor, but it means we will have to throw away the whole sandbox module. This is a pretty high impact as it will change the complete project structure and plan, but it is not enough to end the project.
2. **No VR on GNU/Linux or OS X**
   Because Oculus dropped support for OS X and GNU/Linux in May of 2015, the chance of not somehow getting VR to work in GNU/Linux is pretty high. Our research has shown that there is a possibility we might be able to run our program under WINE, a program that creates a compatibility layer between Windows programs and GNU/Linux, but that itself is a bad workaround. The impact would be medium, but not detrimental, as we could just have a dedicated Windows machine to test with VR on, rather than testing VR on our own machines that might not be able to run an Oculus, anyway.
3. **Not getting VR hardware**
   We are currently not in possession of VR or VR capable hardware, without it the VR part of the project will be taken out and we'll have to make a workaround with a monitor instead. It will not end the project but it will severely impact the project plan. Currently set to low chance as we think Simon McCallum will fix this with his magic.
4. **Work or code being lost**
   All code and assets are being saved in git repositories on Bitbucket, plus on group members own devices, meaning the chance of partial or complete data loss is rather low. Obviously if on the odd chance that all data is lost, it would be severely detrimental to the project and might even end it.
5. **Temporary inaccessibility to Bitbucket**
   Once in a while Bitbucket, which hosts our repositories, will have a couple of minutes or hours with temporary downtime. This will stop us from pushing and pulling from the repository, but it will not stop progress.
6. **Hardware problems, personal or related to project**
   If a part of or a complete device related to the project fails, it will either need to be replaced or fixed, or we will need to change the project plan, optionally pulling features out of the project. If a developers device stops working we will figure out

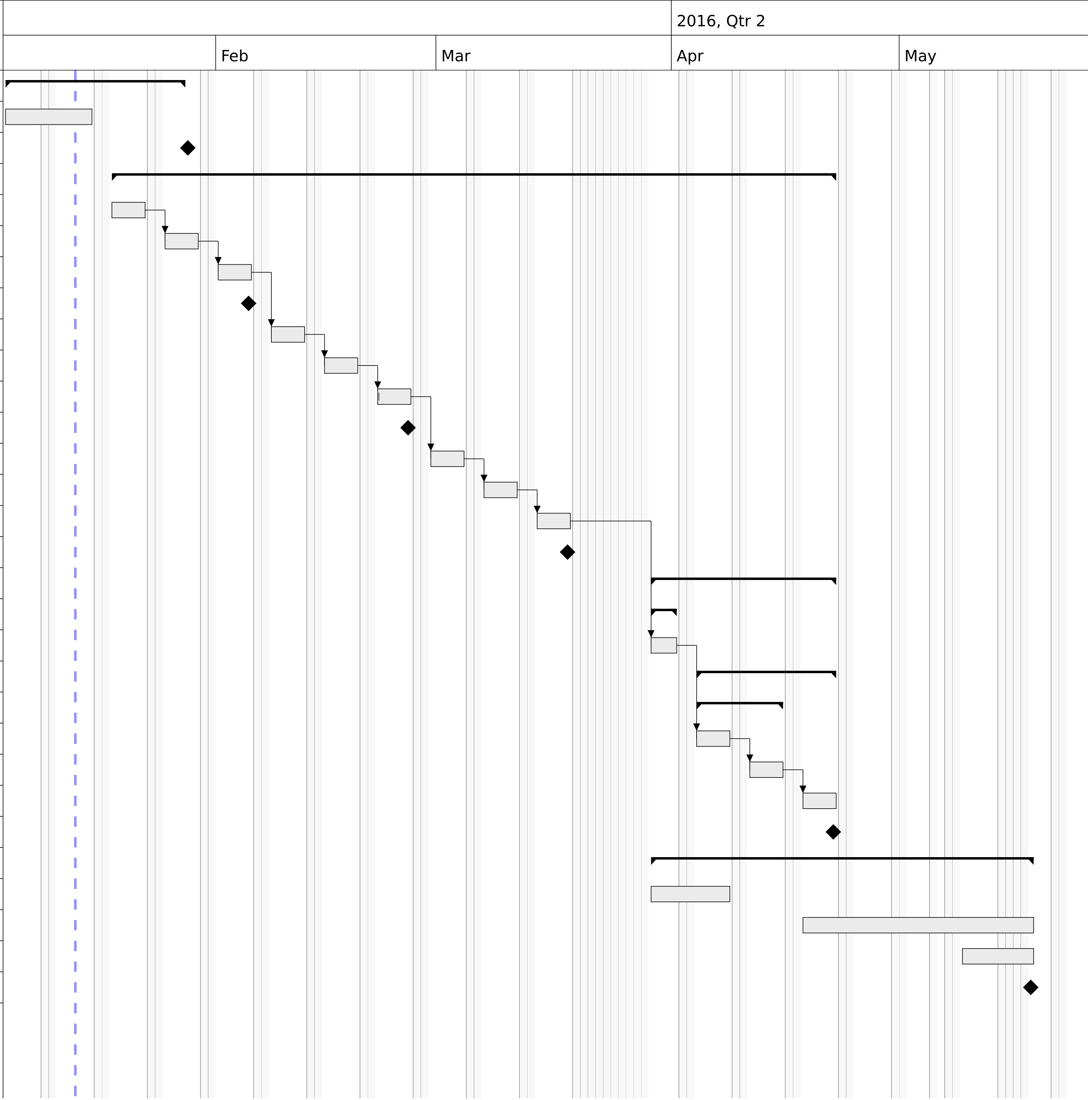
a workaround, possibly using school computers, possibly replacing said device.

7. **Awful look and feel**
   If the look and feel of the experience is awful we will have to do tweaking and testing in order to fix it. The impact is pretty high as it could potentially affect the next risk, risk #8.
8. **VR/Cybersickness**
   There is a pretty good chance that players and users of the project will feel either nauseas or dizzy while wearing a VR headset; they will feel cybersickness. The impact is high as it will affect player experience negatively. The workaround is either that player not using VR, or not playing at all, or there might be a possibility we can fix it somehow.
9. **Too much focus on specific parts of the project**
   Scrum plus our project plan strictly defines when and what we will be working on, but if there is a project change coming from one of the other risks above we might end up focusing on other parts.

B VRterra: Gantt Chart

Original planned Gantt chart for VRterra from the project plan. Actual project differed significantly. We did not have any time for user testing and in sprint 5 we switched from our own engine to using Irrlicht.

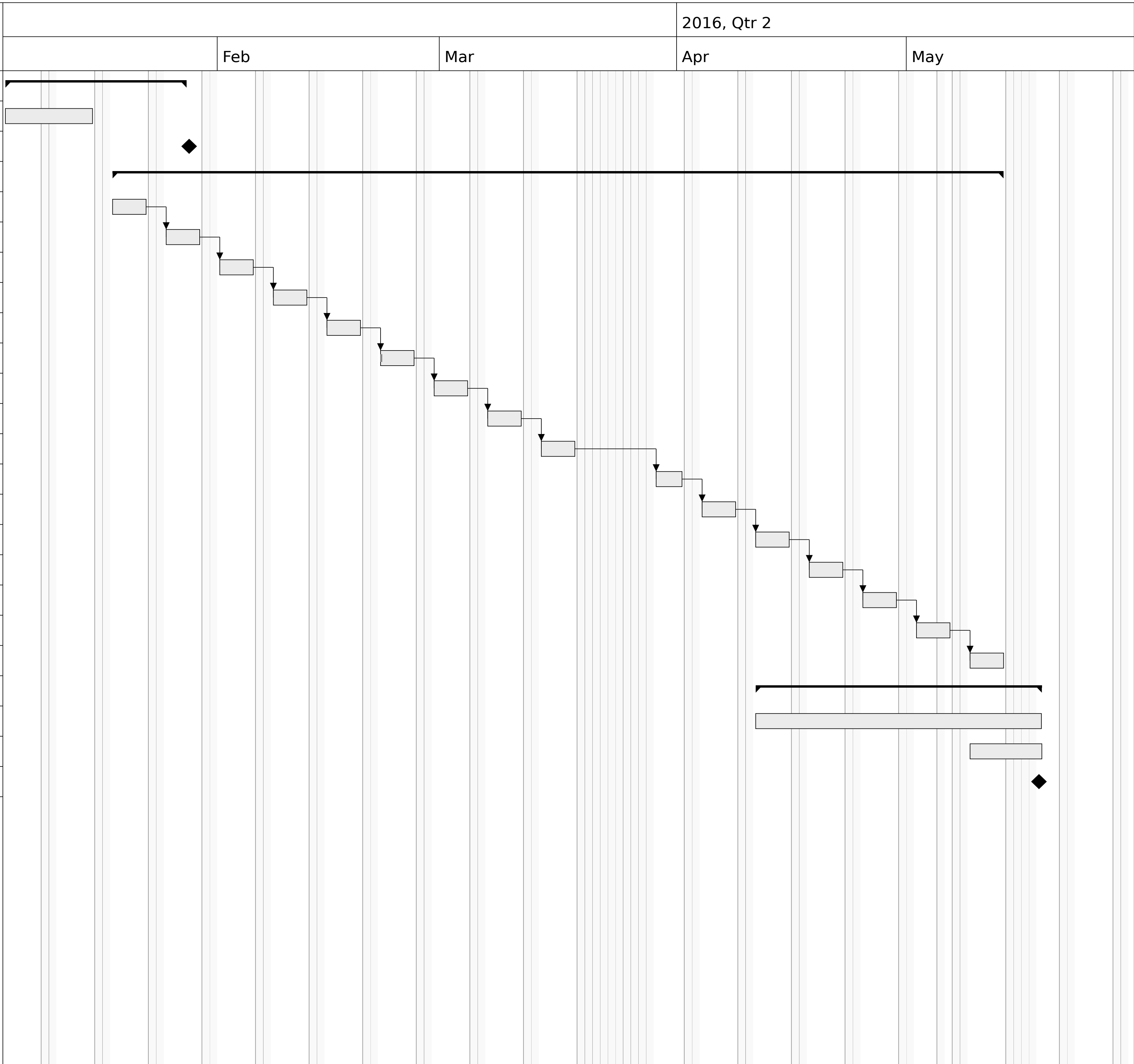
Name	Work
Planning	**10d**
Project planning	10d
Project plan hand in deadline	
Development	**64d**
Sprint 1	5d
Sprint 2	5d
Sprint 3	5d
Terrain Editor & VR Game in Alpha	
Sprint 4	5d
Sprint 5	5d
Sprint 6	5d
Sandbox setup & program	
Sprint 7	5d
Sprint 8	5d
Sprint 9	5d
Beta	
Testing	**19d**
Software analysis (bugs, performan	**4d**
Sprint 10	4d
Issue fixing	**15d**
User testing	**10d**
Sprint 11	5d
Sprint 12	5d
Sprint 13	5d
Testing complete, "Release"	
Documentation	**35d**
System documentation review	9d
Writing report	20d
Report review	6d
Report hand in	



C VRterra: New Gantt Chart

Edited, actual Gantt chart for VRterra from the project plan. Actual project differed significantly.

Name	Work
Planning	**10d**
Project planning	10d
Project plan hand in deadline	
Development	**78d**
Sprint 1	5d
Sprint 2	5d
Sprint 3	5d
Sprint 4	5d
Sprint 5	5d
Sprint 6	5d
Sprint 7	5d
Sprint 8	5d
Sprint 9	5d
Sprint 10	4d
Sprint 11	5d
Sprint 12	5d
Sprint 13	5d
Sprint 14	5d
Sprint 15	4d
Sprint 16	5d
Documentation	**30d 6h**
Writing report	24d 6h
Report review	6d
Report hand in	



D Code Conventions

Mårten Nordheim & Nichlas Severinsen

14.01.2016

D.1 File extensions

All header files will use the .h extension while a class's implementation should generally reside in a .cpp file (the exception being template classes and functions.)

D.2 Header files

To avoid multiple inclusions of a single header file during compilation, each header file should be protected with an include guard, as such:

```
#pragma once
```

D.3 Include statements

Project includes should be at the top. This is then followed by library includes, and finally the standard library includes.

D.4 Naming

Private variables inside of classes will be prefixed by an 'm' and an underscore ('m_'). This is followed by the variable's name in a 'lowerCamelCase' format like such:

```
        float m_value;
```

Classes and namespaces will be named using 'UpperCamelCase'.

```
namespace Foo
{
    /**
    * Description of the class 'ClassName'
    */
    class ClassName
```

Global variables will be prefixed by a 'g' and then an underscore ('g_'). This is followed by the variable's name in a 'lowerCamelCase' format. Although global variables should, preferably, never be used:

```
int g_integer;
```

Functions are named using 'lowerCamelCase'. No prefix or suffix. Every parameter in a function (and constructor) should also be named using 'lowerCamelCase'

```
        void functionName(int myInteger);
```

Constants are named using all caps, with words separated by underscore ('_')

```
const int MAX_AMOUNT;
```

D.5 Commenting

We will be using the Javadoc standard to comment on all classes' and functions' functionality. Beyond this, any other comment will be used to clarify what a piece of code does if it cannot be self-explanatory.

D.6 Formatting

Braces go on a separate line in all cases.

```
namespace Foo
{
    /**
    * Description of the class 'ClassName'
    */
    class ClassName
    {
    public:
        /**
        * Description of the constructor
        * @param value Description for the parameter 'value'
        */
        explicit ClassName(float value);

        /**
        * Description of the destructor
        */
        virtual ~ClassName();

    private:
        /**
        * This is a function
        */
        void functionName(int myInteger);

        float m_value;
    };

    void ClassName::functionName(int myInteger)
    {
        if (myInteger > 42)
        {
            // [...]
        }
        // [...]
    }
}
```

D.7 Pointers

Pointers should be associated with the type, not the variable. It is only semantics, but this way is more logical.

```
// How the different ways can be interpreted

// 'i' is a pointer to an integer, correct way
int* i = nullptr;

// The type is int, but 'i' is only a pointer to one.
// How the compiler reads it, but not the best for readability
int *i = nullptr;

// please, don't
int * i = nullptr;

// although prefer auto when possible
auto something = objectInstance.getPtrToSomething();
```

An indentation is indicated with 4 spaces. Using spaces instead of tabs will make the code look the same no matter what editor you view the code in.

Additionally, there is set up a clang-format-file which will take care of a lot of the formatting. This should be used before committing code to maintain consistency in all code.

E Sprint Reviews

Mårten Nordheim & Nichlas Severinsen

22.01.2016

E.1 Sprint 1

Sprint Goal

1. Set up environments
2. Get thumbs up on environments from Simon
3. Engine & libraries

The goal of sprint 1 was to get our environments set up, as well as getting confirmation from Simon on our environments, we tried sending an email but got no reply. In addition we wanted to start coding and actually create something, but the restriction for what to make was not entirely set, so we settled with getting a base engine and a terrain library done.

Sprint Review

1. Our environments were set up
2. We did not get a reply from Simon
3. Base engine & base terrain library set up, still needs some more work

Sprint Retrospective

Currently we seem to be doing fine, going at a regular pace. We will improve by reading more about rendering and doing more research where applicable. Some terrain library algorithms could use some refactoring.

E.2 Sprint 2

Sprint Goal

1. Get base engine to render
2. Set up terrain editor
3. Networking library to pipe terrain

Sprint Review

1. Got base engine to render
2. Did not set up terrain editor
3. Started, but did not finish networking library

Sprint Retrospective

This week we were really tired because we slept too little, but we finally got the engine to render! The networking library is functional, meaning we can send and receive packets from clients to servers, but we have yet to send a terrain via network. There were some minor hiccups with networking that slowed down progress, mainly because we have never used those libraries (ENet and ProtoBuf) before, which is also why we have not gotten it to pipe terrain.

E.3 Sprint 3

Sprint Goal

1. Update VBO
2. Finish and polish networking
3. Set up terrain editor
4. Config loading for engine

Sprint Review

1. Got VBO to update
2. Config set up
3. Terrain editor "set up" but not in "working" condition
4. Started experimenting with lighting
5. Started experimenting with UI
6. Ray for mouse picking

Sprint Retrospective

We were able to serialize and pipe terrain through networking as well as de-serializing it, but networking is still in early alpha mode and not considered "done". Config is set up and works, some settings need to be globalized in a smart way. Terrain VBO now updates and you can see the changes as they happen in the base-game/app! There has been some work on mouse picking so one can actually edit the terrain with a mouse, but that is far from done. A more advanced shader was added, as well as normal calculation but it seems buggy, unfinished. We can also draw some 2d boxes from a rendercomponent. We are slowing down a bit, but we will improve by sleeping more and doing more research and reading. We did not really reach the first milestone, there is no VR "game" per se, but there is a really really early alpha terrain editor.

E.4 Sprint 4

Sprint Goal

1. Basic networking (client/server) finished
2. Mouse picking needs to get done for terrain editor
3. Refactoring
4. Fix lighting

Sprint Review

This sprint we did a lot of work on the goals, but while talking to and discussing with Simon it seemed a better course than making our own graphics/game engine it would be better to use an already made one, so in the middle of the sprint we started researching more engines. Torque3D was suggested by Simon but neither of us managed to compile it on our machines – Mariusz Nowostawski said he never got it to compile on his Mac, either. We looked into a lot of other different engines and wrote down some quick notes about why or why not use them. At the end of this sprint it seems Irrlicht is an interesting engine, so long we can get the terrain mesh to change in real time.

Sprint Retrospective

We will have to decide on what to do next, in the sprint planning for sprint 5, but in the event we choose to use a new engine we will need to properly read its documentation and make sure it compiles and works as we want it to work – to get better.

E.5 Sprint 5

Sprint Goal

1. Get kinect to work
2. Redecide engine
3. Get working with new engine, plug in already made compatible libraries

Sprint Review

1. Switched engine to Irrlicht
2. Irrlicht does collision
3. Got access to kinect but no video (on windows)
4. Mouse picking works!
5. Networking in testing stage
6. Plugged in some compatible already made libraries.

Sprint Retrospective

Continue doing daily scrum. Start doing static analysis, somehow. Start making better retrospectives, like what to stop and start doing. Start writing down research and ideas we get along the way in the report.

E.6 Sprint 6

Sprint Goal

1. Try the other kinect that Simon has
2. Finish basic/base GUI
3. Merge terrain library with irrlicht terrain

Sprint Review

1. Tried the other kinect (and the first one as well (but not in VMware))
2. Did not have time for GUI or irrlicht terrain
3. Studied some of the SARndbox source code
4. Tried to get the kinect running on a computer running Manjaro (Linux distribution) (no success)
5. Successfully got both kinects working on a Linux Mint live USB.

Sprint Retrospective

We need to get better at planning ahead and estimating how much time each task will take.

Extra Sprint Notes

In this sprint we prioritized getting kinect working on Linux Mint after a talk with Simon earlier in the week. This was done because of how novel the sandbox setup is and it would then make more sense to have our final setup fully compatible with the other sandbox setups that already exist around the world.

E.7 Sprint 7

Sprint Goal

1. Read heightmap using the Kinect 2 library from Oliver Kreylos / libfreenect
2. Figure out how to use the CalibrateProjector application, also from Oliver, and how to apply this output to our own rendering.

Sprint Review

1. Got depth buffer out from kinect device
2. Got irrlicht's terrain to update based on data received on the network. It has some issues however and it needs some cleaning up.
3. Went to Hamar and talked to Ole Andreas, founder of Krillbite, and Trond, programmer at Krillbite.

Sprint Retrospective

This week we went too much back and forth. Nichlas thinks that there is not enough consistency and that the project plan had little to no value. Still need to get better at planning.

E.8 Sprint 8

Sprint Goal

1. Make a basic API for the kinect
2. Integrate the height map from the kinect into a project.
3. VR: Decide on whether to use Unreal/Unity/Irrlicht. Needs some research.

Sprint Review

1. Made a basic API for the kinect
2. Integrated the height map from the kinect
3. Collision and lighting in irrlicht now works
4. Networking sync with mesh so it updates terrain

We are waiting a bit to make the decision about unreal or unity, Simon has not been available this week.

Sprint Retrospective

We have gotten stuck on some things during this sprint, not really our fault in one way.

E.9 Sprint 9

Sprint Goal

1. Refactor out editor, base.
2. Figure out VR on GNU/Linux

3. get VR hardware (oculus)
4. Start on VR game
5. Start looking at calibrating the kinect

Sprint Review

Nichlas was hospitalized for appendicitis. Mårten got TCP and coloring of the terrain done. The rest of these goals we have sort of pushed forward.

Sprint Retro

Be less sick. Be more productive.

E.10 Sprint 10

Sprint Goal

1. Refactor base
2. Projection

Sprint Review

Sprint 10 was cut rather short, we started wednesday (ish, a little bit on tuesday), but we were able to refactor base and set up the project on the computer we are borrowing. There were some struggles with segfaults and kinect not wanting to connect, but it seems we have solved most of these. We set up a projection app to act both as server and to project heightmap back onto sand.

Sprint Retro

Overall decent start after vacation.

E.11 Sprint 11

Sprint Goal

1. Fix coloring
2. Get a sandbox
3. Talk to Simon
4. Focus more on report

Sprint Review

We somewhat did coloring. We also managed to project terrain orthographically, as required for the projection application – but that needs a little more work for now. Simon was nowhere to be found, we did not get a sandbox. Skeleton was made and some of it filled in in the report.

Sprint Retro

Make better retros.

E.12 Sprint 12

Sprint Goal

1. Desperately need a sandbox setup.

2. TCP server needs to be able to receive a position (marker(s))
3. Edit open source VR game to use our terrain

Sprint Review

TCP server has the ability to receive a position for a marker, but more work remains on it (such as getting the marker info to the 'sandbox' application for it to draw.)

Sprint Retro

Mårten did not really count on actually finishing implementing markers on the TCP server in a week. In retrospect having 1-week sprints may have been too short overall for a project like this.

E.13 Sprint 13

Sprint Goal

1. Continue implementing the TCP marker system
2. Let the sandbox receive markers
3. Write more in the report

Sprint Review

The TCP server can now initialize and receive markers. The server-library now gets compiled into the sandbox and the sandbox reads the markers from a singleton class which manages the the markers (MarkerManager.) We also worked a bunch on the report.

Sprint Retro

This week we had less goals and managed to finish them. The goals were also a bit smaller than usual as well. We should also designate who writes what in the report instead of going back and forth and writing wherever we feel like.

E.14 Sprint 14

– extended sprint – Note: We will just keep going with sprints until we have handed in the report; the project plan only includes 13 sprints.

Sprint Goal

1. Adjust the Terrain class's heightmap to not include padding
2. More work on report
3. Calibration

Sprint Review

1. We finished adjusting/removing the padding from terrain
2. We did do more work on report
3. Some calibration was done, allowing us to project pretty accurate on the table

Sprint Retro

Work more with the report.

E.15 Sprint 15

Sprint Goal

1. More work on report
2. User testing
3. More calibration
4. Finish marker position to projection

Sprint Review

1. We worked more on the report
2. We did not do any user testing as the day suggested was a holiday, oops
3. Refactored Base since it became hard to maintain/manage/navigate
4. Minor tweak to the shader

Sprint Retro

We need to work even more on report

E.16 Sprint 16

Sprint Goal

1. Finish the report

Sprint Review

1. We finished the report
2. Tested Unity VR Demo that Johannes helped us set up.
3. Merged into master

This sprint lasted from Monday 9th of May to Wednesday 18th of May, 2016

Sprint Retro

Nothing needs to change

F VRterra: Meetings and Questions

Chronological log of meetings with external people such as the supervisor and employer.

F.1 Temporal record of meetings

04.01.2016 Meeting With Supervisor About Project

Talked with Simon to discuss bachelor project and ended up with VRterra.

27.01.2016 Project

Simon said Krillbite came back and are becoming external contractor, but not oppgave-giver or "owner". Created a signed agreement over the IP of the project, included with the project agreement which we also signed. Discussed some more structure and set up of the project.

10.02.2016

Meeting with Simon, we discussed: Grab the kinect from the game lab and start doing stuff with it. Write up evaluations of engines. Instead of doing mouse picking in terrain editor do a 2D corner map. Simon will look into building a sandbox.

-in between-

We have had some short meetings with Simon every now and then, he has set some requirements like a small API for getting depthmap out of kinect and possibility of getting that data sent over the net. We have basically been updating him on progress back.

02.03.2016 Hamar Game Collective

This day we were at the Hamar Game Collective together with two other bachelors groups and Ole Andreas; the ex-founder of Krillbite, Trond; a programmer at Krillbite. Together with the other VR group all of us talked about our projects, explaining what the plan or concept was, and got some feedback and ideas: We have basically now decided that we are going to make a Racing type of game, as a demo, with transitioning real time terrain input. For now we should focus on getting the kinect to work, smooth terrain transitions and place a player (later; car), possibly with VR. Clean artstyle. Hologram/map in car for driver. Freeroaming. Trond suggested using voxels, we are probably not going to. He also suggested making slopes for jumping to collect some sort of item in the air.

11.03.2016

Quick Friday meeting with Simon: If protobuf -> make unity "hello world" example/template discuss sending terrain data, either with protobuf or json or xml, or something, compare and write about it. make a diagram of the flow of data, should have description (especially of the networking, not explicitly). Do a webserver that logs saved heightmaps, saving it as a file/image is a nice way of doing it no need to implement it, but do describe it, low priority - but cool feature to have. write about it!

08.04.2016

Quick Friday meeting with Simon: Vitensenteret will be making the sandbox Projector needs to get a texture and id/pos and the projector renders that at that pos. or something like that. _Could_ expand by saying which textures /colors the projector is using etc. look for a game with VR, capable of handling our networking. linux networking needs to get and be sent a texture

he would like to see a very simple first person viewer, basically the

minimum: terrain from kinect, send to program that displays a heightmap - put player - then the position of the viewer is sent back to the server and the player pos is displayed on the sand.

19.04.2016

Today we had a meeting with the Vitensenteret here in Gjøvik. We pitched the project and they asked for the requirements, which we gave them; They needed a computer, a kinect, a projector and a sandbox – we would set the project up for them if they built the hardware, and Simon also wanted a sandbox for the school. We could potentially provide one of the kinects we had.

- in between -

The following month from last meeting we have had several short and medium meeting with Simon, discussing solutions to problems and such. We have also moved our location and equipment from Mustad to the game lab in the A building, room 253.

10.05.2016

This day Simon came in in the morning and got us to move to the room where the Oculus and the Vive is, discussed some solutions to problems, some report, and got Johannes to come help us with the VR stuff with Unity.

G VRterra: Work Logs

Work logs in chronological order. Varying amount of detail, entries without specified time can be assumed to have lasted from 08:00 to 15:00. We also have time logs from Toggl.

04.01.2016 Bachelor Information Meeting

Met up 08:00 to think on choosing/creating a project. Later talked with Simon to discuss this and ended up with VRterra. Ended 15:00

05.01.2016

08:00 to about 15:00. A lot of thoughts about the project, set up BitBucket repository for the application (this was later moved under the VRterra team). Decided on using Scrum and started researching and testing many different tools.

06.01.2016

Intensive testing of different tools.

08.01.2016

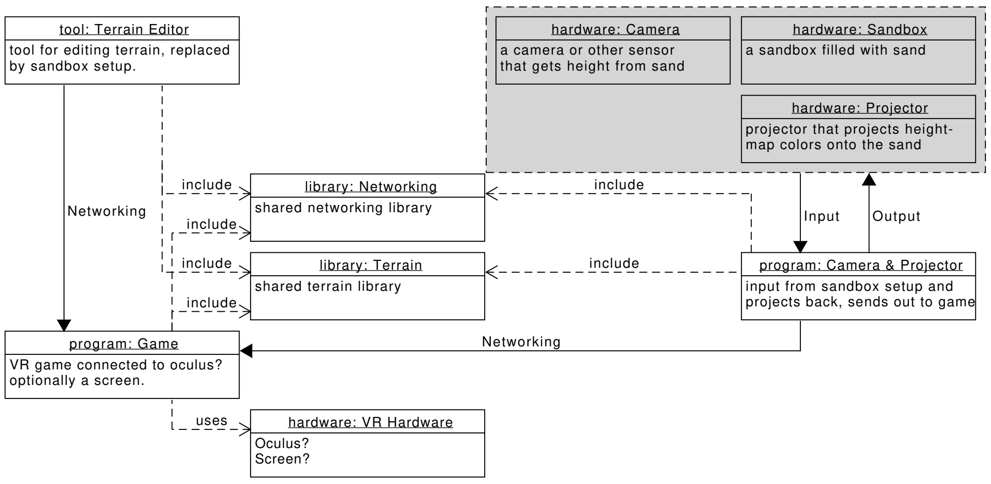
More intensive testing of different tools.

11.01.2016

Official start of school, met around 08:00 at the GameLab then went for a crash course in the bachelor project from 10:00 to 12:00, and back to the crammed GameLab until around 15:00. Sent an email to Simon as he was pretty busy that day with some questions about the bachelor project.

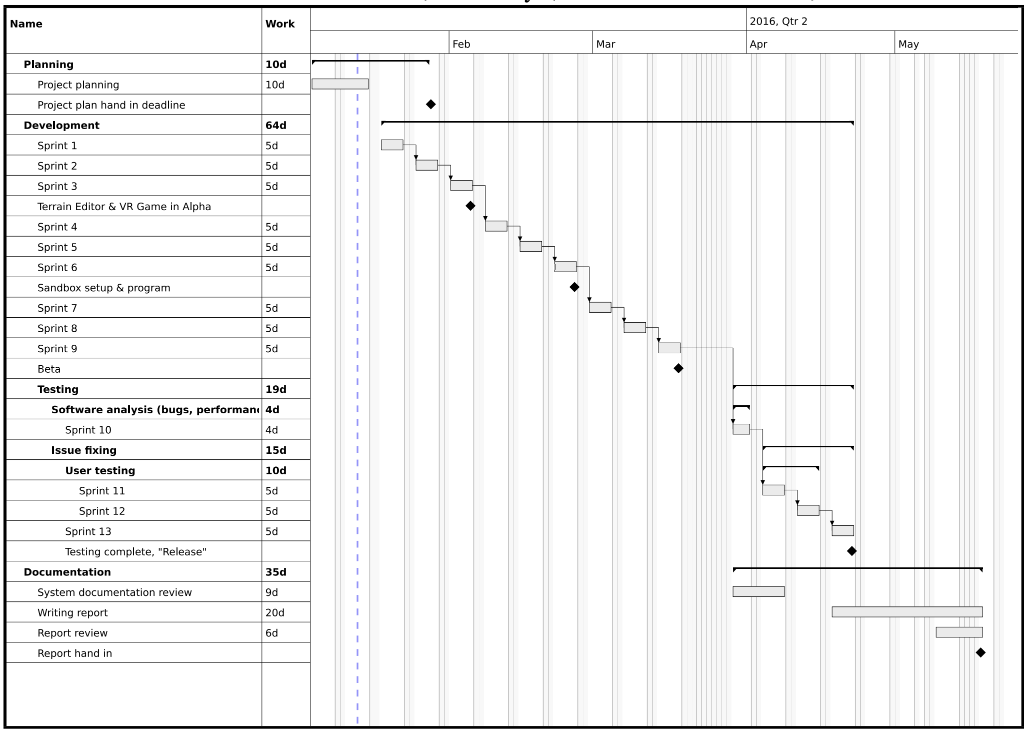
12.01.2016

VRterra team set up on BitBucket with a repo for the report and the application(s). Started editing the report template to our liking, filling in previous days. Some more research on tools and libraries to use. Got some of the stuff we drew down on paper into the report. Researched UML tools to use (sigh), ended with umlet. Researched a ton of gantt diagram tools.


tool: Terrain Editor
tool for editing terrain, replaced by sandbox setup.
hardware: Camera
a camera or other sensor that gets height from sand
hardware: Sandbox
a sandbox filled with sand
hardware: Projector
projector that projects height-map colors onto the sand
Networking
include
library: Networking
shared networking library
include
include
library: Terrain
shared terrain library
include
include
Input
Output
program: Camera & Projector
input from sandbox setup and projects back, sends out to game
include
program: Game
VR game connected to oculus? optionally a screen.
Networking
uses
hardware: VR Hardware
Oculus?
Screen?

13.01.2016

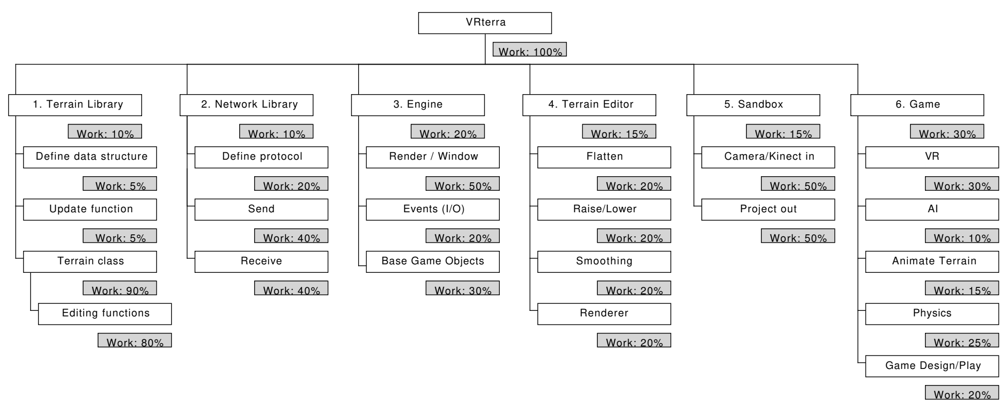
Nichlas: Filling in project plan and planning. More tool research found Planner and used that to create a gantt diagram. Mårten: Setting up projects + structure, build systems, google test, oculus vr library/sdk + sdl + opengl. Decided on using C++14 + OpenGL + SDL + Oculus VR libraries, rather than using a separate engine like Unreal (because of no real-time terrain modifications) or Unity (because C#/not C++).



Name	Work
Planning	**10d**
Project planning	10d
Project plan hand in deadline	
Development	**64d**
Sprint 1	5d
Sprint 2	5d
Sprint 3	5d
Terrain Editor & VR Game in Alpha	
Sprint 4	5d
Sprint 5	5d
Sprint 6	5d
Sandbox setup & program	
Sprint 7	5d
Sprint 8	5d
Sprint 9	5d
Beta	
Testing	**19d**
Software analysis (bugs, performanc	**4d**
Sprint 10	4d
Issue fixing	**15d**
User testing	**10d**
Sprint 11	5d
Sprint 12	5d
Sprint 13	5d
Testing complete, "Release"	
Documentation	**35d**
System documentation review	9d
Writing report	20d
Report review	6d
Report hand in	

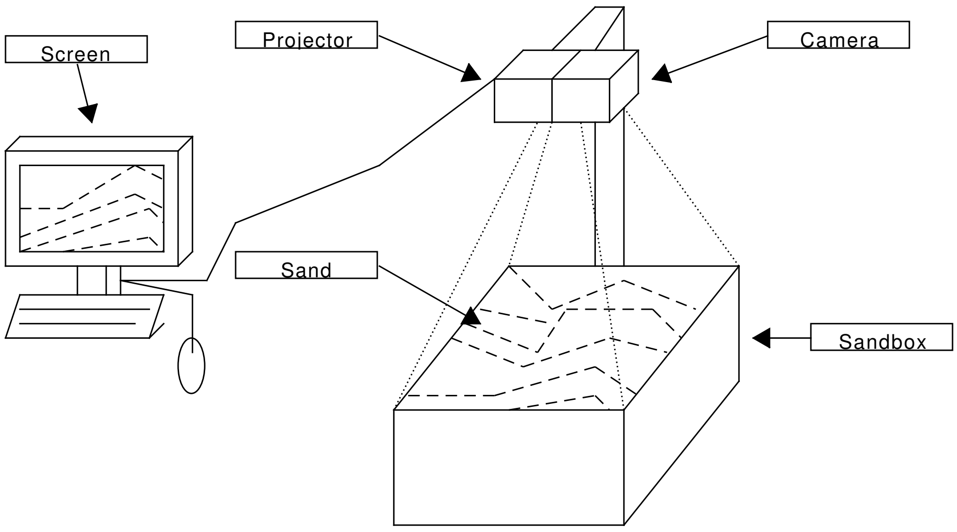
14.01.2016

Nichlas: WBS, GDD, Project plan, suggested game flow diagram. Mårten: library research, wbs, research, project plan, code convention,



15.01.2016

Project plan, visualization, Risk analysis, Background, Code convention.



18.01.2016

Worked on planning and creating a terrain library, looked at possibly using the ogre game engine and their terrain.

19.01.2016

Yesterdays research showed that plugging in a game engine like OGRE, Irrlicht, or Panda3D, would be difficult considering that we need to change the terrain during runtime. Today we focused on setting up our environment and a small engine ourselves, instead of using an already made game engine. Research, terrain library. GSL. Window up and running.

20.01.2016

More engine setup stuff, architecture, more terrain library.

21.01.2016

Today we kept going with the stuff we did yesterday, terrain library and base engine.

22.01.2016

Last day of sprint 1; Setup, Engine & Libraries. We worked some more on those and did sprint planning, sprint review meeting and sprint retrospective.

25.01.2016

First day of sprint 2; got rendering to work, did some research and tried setting up networking. Chose to use ENet.

26.01.2016

More refactoring, rendering, engine, network.

27.01.2016

Today we had a meeting with Simon at 09:00, but due to misunderstandings we started around 9:20. Check meeting log for the same day for more info. Mårten did some more work on the basic engine, rendering, tests, scenes and such. Nichlas finished up, proof-read and polished the project plan, as well as update the WBS with more details, then went on to making a network protocol with protobuf.

28.01.2016

Networking and Engine (rendering). Server/Client works! Rendering works!

29.01.2016

End of sprint 2; Networking and Engine. Sprint review, retrospective and planning.

01.02.2016

Mårten: I added the Config class which loads files stored in json which we use as configuration files in the project. After this I made it so that the buffer object is updated when the heightmap is altered. Then I spent the rest of the day trying to get ENet to work properly on Windows.
Nichlas: Spent some time trying to get terrain sent over networking but gave up, Mårten took over networking. I then proceeded to try implementing normals and simple lighting.

02.02.2016

Mårten: I made ENet work properly on Windows after some more trying. The order in which the CMake include_directories statements were in seemed to make a difference. After this I worked some on the protobuf+ENet stack we have to get it to transmit anything at all. Which I managed to do towards the end of the day.
Nichlas: Normals and Lighting "implemented" but they seem buggy, rewrote a shader Mårten had from before to fit our project. Researched doing mouse picking with ray casting.

03.02.2016

Mårten: Today I changed parts of the protobuf protocol we had, to allow for smaller packets to be sent when the terrain is updated. I also started on a Server-class which is

going to run independently and will distribute these changes to the clients. This might change, but for now it will stay. Sadly, I discovered that the project now doesn't compile properly in Release mode for some reason. I tried to solve it but after a long time I came to the conclusion it wasn't critical enough to keep working on right now and wrote it down as an issue.
Nichlas: More research on mouse pick raycasting but got nowhere with it so I started doing some simply (G)UI for Terrain Editor, as well as minor changes to terrain editor.

04.02.2016

Mårten: Today I worked more with the Server-class. It was mostly refactoring and moving functionality around. I also made another, smaller, change to the protobuf file. I didn't get to test if it works correctly today, so I will have to check that tomorrow.
Nichlas: (G)UI stuff, figured out glOrtho (I think) and made it possible to draw colored squares as UI.

05.02.2016

End of sprint 3
Nichlas: Back on trying to figure out mouse pick raycasting so we can actually edit in terrain editor.
Mårten: Did some work on a base class for client. This may end up being promoted to the "only" client class, but it might prove to be useful to overload a basic one. Nothing otherwise noteworthy was worked on.

08.02.2016

Mårten: Since we have not been able to finish up the networking yet I have still been working on that. Sending and receiving works fine but integrating it with the terrain in a sensible manner (preferably not send too many packets in a too short amount of time while also not sending them too rarely) is taking a little longer than originally thought. Combined with wanting to have a clean interface on both networking and terrain and it gets complicated. It may be better to just get it done and clean up after.
Nichlas: Working on; Normals, Lighting, Shaders, Mouse picking.

09.02.2016

Mårten: Updated the protobuf protocol again, made it simpler to both save time and sanity. I will have to optimize it later if it becomes a problem. Was finally able to have some terrain data transmitting between a client and a server, and saw the terrain being changed. There may be a bug somewhere as it does not show up the way the code suggests it should. Will have to look into that. Also added a way to tell the difference between a server, a client and a terrain editor/sandbox in the packet, probably only necessary in the first packet, however.
Nichlas: Normals, Lighting, Shaders, Mouse picking.

10.02.2016

Nichlas: Today we had a short meeting with Simon, check the meeting log for this day to see what was mentioned. We then went ahead and tested and set up TeamCity and Jenkins. We also look into others like Travis-CI and Strider.
Mårten: Worked more on networking, little noteworthy progress today. We also looked

into getting some continuous integration going, but it has proved difficult. We also looked at Torque3D which we will likely look more at and possibly start using.

11.02.2016

Mårten: Today I looked at Torque3D a whole lot and attempted to make it compile so we could test its terrain-capabilities. Unfortunately I have not yet been able to test it and will have to try again tomorrow.
Nichlas: Testing out more 3D engines, tried getting Kinect 360 to work on GNU/Linux but to no avail.

12.02.2016

Mårten: I spent most of this day trying to the Torque3D to compile, again. And, again, it refused to compile no matter what I did. We're thinking to drop it because of this.
Nichlas: Irrlicht research, testing, tutorials, and it works! Review, Retro, Planning.

15.02.2016

Start of sprint 5.
Mårten: Looked at terrain in Irrlicht. Got a small piece to change in real-time.
Nichlas: Irrlicht, research, tutorials, testing.

16.02.2016

Mårten: Wrote a module for cmake to easily include and link the Irrlicht engine. After this I did some other smaller changes to the new project setup, and brought over and started integrating some of the code from the project we were working on earlier.
Nichlas: Irrlicht, research, tutorials, testing.

17.02.2016

Nichlas: Irrlicht, refactored some base irrlicht code into a base class.
Mårten: Started the day by working a little bit on a compilation issue we had at the end of the previous day. After this I went back to work on the networking. I performed a smaller refactoring and added some checks, some of which caught bugs which could've been hard to track down had they been encountered later.

18.02.2016

Mårten: Some refactoring and fixed one issue in the networking part. Then went on to trying to get the kinect up and running on Windows.
Nichlas: Started working on a Base GUI class that both Game and Terrain editor can use.

19.02.2016

Nichlas: End of sprint 5. Did the irrlicht GUI tutorial to gain more insight in how to set up GUI, thought and got stuck at how to set it all up in a comfortable way. Retro, review, planning.
Mårten: I worked more on getting kinect working. I eventually got the drivers to work properly for libfreenect (which we would have to be using because of the cross-platform requirement.) The motor is working fine but I am not getting any video feed out of it, so this kinect may be faulty. Will try with another one next week.

22.02.2016

Start of sprint 6
Mårten: I fetched the "Kinect for Windows" and checked that it works. It did. We had an ongoing conversation with Simon on Skype throughout the day about the project. The main takeaway from that at this moment is that we're targetting Linux as our main platform in regards to the sandbox. I then spent some time setting up my Linux virtual machine environment to be able to compile and work with the SARndbox source code.
Nichlas: More Irrlicht GUI and tutorials.

23.02.2016

Mårten: Today I spent most of the day looking through the SARndbox source code: compiling and running some of the test applications it (and its underlying dependencies) produces to see that everything was working correctly. After a while I got it to compile properly and run consistently. There is some strange artifacts on the video feed, however, and I will have to look into it. It may, hopefully, be a side effect of using a virtual machine as opposed to a machine running directly on the hardware.
Nichlas: Irrlicht GUI!

24.02.2016

Today we looked further into the strange kinect video feed. We tried solving it in a few different ways, mostly attempting to run it on Nichlas's computer. For some reason we were unable to get the kinect properly connected and decided it may have something to do with him using an Arch-based OS while the official instructions specifically name Mint/Ubuntu/Fedora. We are now attempting to see if it will run on Mint on the same computer, but we will get back to this tomorrow. Had a status report with Simon.

25.02.2016

Linux worked on getting Kinect to work in Nichlas Mint. Mårten studied more code to get a better understanding of how the SARndbox application works. At the end of the day we both went to pick up a desktop and a projector to work with on the project.

26.02.2016

End of sprint 6.
Today we worked on getting mint installed on the desktop. It has a RAID configuration which seems to be causing trouble for the installer. We also wrote the standard end-of-sprint entries.

29.02.2016

Start of sprint 7
Mårten: I read a bit more of the code from SARndbox, but decided to put it off for a little bit since I felt like I wasn't getting anything done. Then I started my attempt at updating the terrain based on incoming packets on the network.
Nichlas: I read a bit more about Irrlicht and Mårten suggested coloring the terrain based on height, so I scratched my head trying to figure it out the whole day.

01.03.2016

Mårten: Continued my attempt at updating the terrain over the network. A little progress but encountered some difficulties in compiling which wasted some time due to silly mistakes (refactoring a class into a library and then forgetting to link it.)
Nichlas: Continued my attempt at updating the terrain color. A little progress but encountered some difficulties with ITerrainSceneNode which wasted some time due to not understading why .Color didn't set the color. (yesterday too). I also did tutorial 23 from the official Irrlicht documentation to try and understand meshes better.

02.03.2016

Mårten: Today I continued trying to get the terrain to update based on the network data. However we did not work for long as we went to Hamar at around 11 to get some input on our project from Ole Andreas and Trond from Krillbite.
Nichlas: Looked at refactoring to get heightmap color to work, again. After a quick meeting with Simon we went to Hamar.

03.03.2016

Nichlas: Finished work on terrain mesh, ran into segfault problems. Tried debugging segfault problems to no avail and ended up making a terrainheightmap class to test some more. Then I gave up and started looking into reading heigthmap from the kinect.
Mårten: Today I went back to doing terrain updates. I did manage to make some of it update, although not all of the terrain was updated when it was supposed to.

04.03.2016

End of sprint 7
Mårten: Today I got the terrain updates working. There are some issues I have to look into, but at least it works now.
Nichlas: after some research I figured it'd be easier to use libfreenect for reading kinect data, and so I looked into that and found out that all we needed was the depthbuffer. Then proceeded to set up a small Kinect.h class to use libfreenect etc. Had some dependency problems. We also figured out how to use the kinect on different linux distros, seems the content in /sys/module/usbcore/parameters/autosuspend file has to be set to -1 so it doesnt suspend the device.

07.03.2016

Start of sprint 8
Today Mårten finished heightmap to Irrlicht mesh sync and then we both looked at the Kinect depth output. Which turned out to be in color and not grayscale as we originally thought. Now we need to find a function to turn the color image into a heightmap.

08.03.2016

Nichlas: Research, cleaned up the Kinect.h and kinect test. Wrote some in the report.
Mårten: Cleaned up and committed the work I did on synchronizing Irrlicht-mesh and Terrain. After finishing this I spent the rest of the day researching VR in Unreal/Unity and attempting to find any info on whether or not they support VR on Linux.

09.03.2016

Nichlas: Tried getting a quick example up and running that pulls data from the kinect into a heightmap and renders it. Had a lot of problems with the ITerrainSceneNode in irrlicht, normals and colors for example so I'll need to try with IMesh instead. We also went to the "Lynkurs"-course no. 2.
Mårten: Read a little more about VR, specifically in Unity, and with regards to Linux, to see if anyone had any extra knowledge, but did not find any more info on it. Started a small refactoring of the synchronization between Irrlicht-terrain and our Terrain class/library since the current implementation requires you to recalculate all of the normals on the terrain every time a single vertex is updated. After refactoring is done it will only be done once at the end.

10.03.2016

Nichlas: Still struggling with properly rendering the kinect data in irrlicht (the whole day), no problems putting it into a heightmap but we need certain features that ITerrainSceneNode does not offer, prompting us to use IMesh or SMesh instead – which says that we have too many vertices.
Mårten: I discovered a small issue regarding the syncing not completely updating all of the Irrlicht terrain. So I tried fixing this. Unsuccessful.

11.03.2016

End of sprint 8
Nichlas: After struggling with rendering for a while, Mårten found out you could call on "setScale" in TerrainSceneNode and it would also calculate new normals, intuitive. Using that I was able to properly do lighting and rendering the kinect data. One of the problems I ran into was TerrainSceneNode always being a square, we're gonna have to cut the remaining part off somehow. Coloring still doesn't work so we'll either have to figure out that, or maybe we could use textures instead.
Mårten: Today I fixed the issue I discovered yesterday, and now the entire terrain gets processed. I also figured out how to recalculate the lighting and the collision on the terrain, which was not updated earlier.

14.03.2016

Start of sprint 9
Mårten: Refactored a little bit and got started on a shader for the terrain. Started with a basic one to acclimatize myself to how Irrlicht deals with shaders, but I need to do more work/reading in this area.
Nichlas was sick and had to leave for today.

15.03.2016

Mårten: I worked from home today as Nichlas was still not feeling well. Tried to work more on shaders in Irrlicht and will likely keep doing so tomorrow as I made little progress on it (Irrlicht being a high-level library makes it harder to get certain things I want sent into the shader.)

16.03.2016

Mårten: I looked at different TCP libraries for C++ with the specific requirement of it being small and standalone. Ended up picking asio and started some work with that.
Nichlas: Hospitalized for Appendicitis.

17.03.2016

Mårten: Continued working on the TCP server-bit and decided that using JSON would be a fairly easy way to serialize the required data. Got stuck with the C++ JSON-library's templates having trouble figuring out what my array of floats was. Will figure out that tomorrow.
Nichlas: Hospitalized for Appendicitis.

18.03.2016

End of sprint 9
Mårten: Worked more on the TCP server bit. Turns out the template issue was not really an issue (it runs like expected) but visual studio still marks it with a red squiggly. And started redoing heightmap access in Terrain to copy-on-write to avoid threading issues which will occur on at least the server (TCP + enet server running in separate threads.)
Nichlas: Hospitalized for Appendicitis.

30.03.2016

Start of sprint 10
Mårten: Refactored some in the branch I'm working in. Encountered a crash which I tried to fix, but will have to continue later.
Nichlas: Worked on projector projection, more research, some refactoring, some planning. The terrain heightmap in irrlicht must be square, so we're planning to cut off the unused part of the heightmap by calculating a top down view that only looks at the parts we're using, to project that out from the projector. We did plan to refactor base out, but as of now it seems we need it for multiple programs so we need to refactor it some more.

31.03.2016

Mårten: Made it faster for kinect to update the terrain's heightmap. And got stuck troubleshooting a threading problem which only occurs on Linux for some reason.
Nichlas: Some more irrlicht-projector, made it get data from the kinect via another thread continously and merged in Mårtens refactoring, plus my kinect branch into the projector branch. We did however run into a segfault problem where irrlicht OnAnimate crashes to desktop.

01.04.2016

End of sprint 10
Today we set up the project on the computer we are borrowing/using for the project, fixed yesterdays segfault by adding mutexes, and focused some on making coloring look okay.

04.04.2016

Start of sprint 11
Nichlas: Research, removed parts of the report that came from Simons project on GitHub – particularily the ones that showed examples of how to do stuff, and uncommented descriptions he had written for what went in what parts. In addition I made a skeleton of sections and subsections (partly stealing from other older theses) and wrote in some of them.
Mårten: Spent more time on the color shader, trying to make it look nice. I also noticed an issue which arose only when the client was connected to a server, which was a segfault on client side and unresponsiveness on the server-side. The segfault was due to a race condition, so I fixed that with a mutex and the unresponsiveness was due to it being slow, so I optimized it.

05.04.2016

Mårten: Related to yesterday, another issue caused the server to become unresponsive so I optimized this today. Discovered a couple of memory leaks (one in the client and one in the server), which exhausted RAM fairly quickly. It was easy enough to fix as it turns out I had forgotten/accidentally removed a single deallocation in both cases. After doing this I did some general clean-up.
Nichlas: Research, filled in more of the parts of the report that I created as a skeleton yesterday. I'm a little bit unsure of how or what to write where, how we're going to structure the document etc. I guess we'll have to get Simon to glance over it (somehow). We also need to have a meeting with Simon about the project.

06.04.2016

Mårten: Further improved transmission of terrain. Used to have a system where it only sent deltas but the Kinect seems to record a lot of noise so it ended up seeing most of the image as changed. This causes the overhead that the delta-system introduced to become too expensive (creating packages up to 3x as large as simply sending the entire terrain again.) For now we solved it by simply retransmitting the entire terrain. Also noticed that despite the Kinect's resolution being 640x480 it sends 8 blank columns, which makes its actual resolution 632x480. Solved by ignored the 8 blank columns.
Nichlas: Wrote more about our development process in the report, more skeleton here and there. Filled in, moved, and changed some text at places.

07.04.2016

Mårten: Wrote some tests and then we both got stuck with a rendering problem in irrlicht which is still unsolved.
Nichlas: Moved Kinect.h into its own directory and added it as library into the CMakeLists that uses it. Then proceeded to get stuck with rendering for the projector, like Mårten said. I personally think we should just draw a 2D image with something like setPixel.

08.04.2016

End of sprint 11
Nichlas: made a new branch game meant for the game part of the project – also the vr part I suppose, branched out from #42-terrain-mesh-sync and merged in the kinect

branch. Then I refactored game main.cpp to use base instead of doing its own thing.
Mårten: Tried to fix our issue from yesterday, although it's becoming more of a hassle than it should be at this point which is a shame. I might look at coding up a small and fast alternative during the weekend.

11.04.2016

Start of sprint 12
Mårten: Was mostly thinking of how to implement new features on the TCP server (receiving coordinates and then relaying that to the other parts of the server), and ended up not doing as much coding.
Nichlas: Researched and trying to find an open source game we can use, as Simon instructed us to. I tried churning through this task by using multiple search engines and the search features on several git repository hosting sites; gitlab, github, bitbucket. To no avail. There were a few candidates that seemed to have what we needed, but most were too undocumented or unfinished they didn't even include a comprehensive "README". If none of those projects found work, we'll most likely have to make our own small VR mock up with something like unity or unreal, depending on what's fastest.

12.04.2016

Mårten: Started implementing/expanding the TCP features. Very rough implementation so far, some issues, need to make it work and then I will refactor it to become easier to work with.
Nichlas: Downloaded and built Unity (took a while), searched for open source/public domain car games/simulators made in unity after realizing we could use Unity's VR feature for the VR game. After hours of searching, finally found one where the car actually moved. Made a new repo with said game (vr-game), tweaked some, found a script that loads texture into heightmap and tried loading a picture that Mårten had taken earlier using the c-sharp_tcp_test program.

13.04.2016

Mårten: Continued implementation of the TCP features. Asio seems like a nice library to work with so far, even though the documentation is lacklustre.
Nichlas: Fiddled around with unity all day, was able to make a script that updated the terrain. Wasn't able to figure out if it works realtime or how to recalculate the normals, though.

14.04.2016

Mårten: Continued working on implementing the TCP features.

15.04.2016

End of sprint 12
Mårten: Continued work on the TCP features and wrote a simple implementation of the Marker and MarkerManager classes. Marker in this case refers to markers to be projected onto the sand.

18.04.2016

Start of sprint 13
Mårten: Made MarkerManager thread-safe since it was going to be accessed by multiple threads (at least on the server.) Refactored a bit, continued work on the tcp server and coded a little more on the C# client-test to easily test and see if the tcp server generates, sends out and reloads the IDs it should. I also wrote sprint review and retrospective for last week as well as the planning for this week.

19.04.2016

Meeting with Vitensenteret at 10:00, check meetinglog for this day.
Mårten: Wrote some on the report before we headed down to Vitensenteret. After we got back I spent some time on the C# library (most of the time was spent rewriting most of it to .NET framework 2.0 since that is what Unity uses.)
Nichlas: Filling in the report

20.04.2016

Mårten: The application can now draw in a simple red circle as a marker whose position can be updated by whichever client requested its creation. In reality it is not that robust (no permission system), but it works.
Nichlas: More report; finally adding bibliography, introduction seems nearly finished – might need some more based on the "minimum 4 pages intro"-thing.

21.04.2016

Today we worked on the report.

22.04.2016

End of sprint 13
Nichlas: More report stuff and fixes. Emailed radx64 about permission for CarSimUnity, and he gave us permission. Used CPack to generate .deb files, these need to be tested more though. Might also want to make .rpm files.
Mårten: Spent more time on the report, wrote sprint review and retrospective

25.04.2016

Start of sprint 14
Mårten: Merging together branches to make the projector run properly. The merging mostly took a long time because it merged unsuccessfully and had to be manually stitched together in many locations.
Nichlas: Worked on report.
We also moved from Mustad into the GameLab since it was a better location to work a test-sandbox setup.

26.04.2016

Mårten: Removed padding from our Terrain heightmap which we had to deal with the differences in the resolution of the Kinect's camera and Irrlicht's terrain (which has to be square.) Also wrote a little bit in the report and a bit of pair programming.
Nichlas: Tried implementing Simon's requested texture mapping in Irrlicht. Came to the

point where we are supposed to have the cutout of the texture.

27.04.2016

Mårten: Worked on the report and tried getting the texture mapping to work in Irrlicht.
Nichlas: Texture mapping.

28.04.2016

Today we got the texture mapping to work, it's not perfect but it works.
Nichlas: Texture mapping, some report
Mårten: Custom mesh to map the texture to, and worked a bit on the texture mapping as well.

29.04.2016

End of sprint 14
Today we did some more work on the shader and the report.
Nichlas: Report
Mårten: The custom mesh now fills the screen when visible. Application now draws topography lines on the terrain.

02.05.2016

Start of sprint 15
Nichlas: Report
Mårten: Refactored Base into Projector and Game.

03.05.2016

Nichlas: Report
Mårten: Spent a little time wrapping up the refactoring and fixing a small bug that cropped up. Also found a bug in the server which causes it to crash rarely when a client disconnects due to simultaneous access to a container. Spent a lot of time thinking about how to extract the selected part of the terrain to transmit over the network as opposed to transmitting the entire image seen by the kinect.

04.05.2016

Mårten: Spent more time thinking about how to extract the selected part of the terrain. It is proving to be difficult to get a proper grasp on what needs to be done. Wrote a dead simple shader to color the terrain based on it's height to use for a method of finding the tallest and lowest point in the terrain.
Nichlas: Report

06.05.2016

End of sprint 15
Today we both worked on the report

09.05.2016

Start of sprint 16 Nichlas: Report.
Mårten: Averaging suddenly worked in the kinect. Merged a whole bunch of things to develop. Fix topography lines filling out a larger area. Worked on report.

10.05.2016

Mårten: Worked on getting some basic VR demo running with Nichlas. Then I worked on extracting a selected piece of the heightmap into a separate heightmap.

Nichlas: Report. Unity, testing VR.

11.05.2016

Nichlas: Report.

Mårten: Made the work I did yesterday compile on Linux. Wrote in the report.

12.05.2016

Mårten & Nichlas: Report.

13.05.2016

Mårten & Nichlas: Report.

14.05.2016

Mårten: Report.

15.05.2016

Mårten & Nichlas: Report.

16.05.2016

Mårten & Nichlas: Report.

17.05.2016

Mårten: Report.

H Project Agreement (Norwegian)

NTNU
Norges Teknisk-Naturvitenskapelige Universitet
NTNU i Gjøvik, Avd. Informatikk og Medieteknikk

PROSJEKTAVTALE

mellom NTNU v/Avd. Informatikk og Medieteknikk (NTNU/AIMT) (utdanningsinstitusjon), og

VRterra (Simon McCallum, Mårten Nordheim, Nichlas Severinsen)

(oppdragsgiver), og

Nichlas Severinsen

Mårten Nordheim

(student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra Jan til June.

   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der AIMT yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra AIMT å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon/fax, reiser og nødvendig overnatting på steder langt fra Gjøvik/AIMT. Studentene dekker utgifter for ferdigstillelse av prosjektmateriell.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. AIMT står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av faglærer/veileder og sensor (intern og ekstern sensor). Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Alle bacheloroppgaver som ikke er klausulert og hvor forfatteren(e) har gitt sitt samtykke til publisering, kan gjøres tilgjengelig via NTNUs institusjonelle arkiv hvis de har skriftlig karakter A, B eller C.

   Tilgjengeliggjøring i det åpen arkivet forutsetter avtale om delvis overdragelse av opphavsrett, se «avtale om publisering» (jfr Lov om opphavsrett). Oppdragsgiver og veileder godtar slik offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og dekan om de i løpet av prosjektet endrer syn på slik offentliggjøring.

Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode mv. som inngår som del av eller vedlegg til besvarelsen, kan vederlagsfritt benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av AIMT til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved NTNU/AIMT og/eller studenter har interesser.

6. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

7. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

8. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i Fronter. I tillegg leveres et eksemplar til oppdragsgiver.

9. Denne avtalen utferdiges med et eksemplar til hver av partene. På vegne av AIMT er det dekan/prodekan som godkjenner avtalen.

10. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og AIMT som regulerer nærmere forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene. Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale med oppdragsgiver, skjer dette uten AIMT som partner.

11. Når NTNU/AIMT også opptrer som oppdragsgiver, trer NTNU/AIMT inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

12. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene i mellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

13. Deltakende personer ved prosjektgjennomføringen:

NTNU/AIMTs veileder (navn): simon.mccallum@ntnu.no Simon McCallum

Oppdragsgivers kontaktperson (navn): Simon McCallum

Student(er) (signatur): Nicklas Severinsen dato 27.01.2016
Mårten Nordheim dato 27.01.2016
dato
dato

Oppdragsgiver (signatur): [illegible] dato 27.01.2016

Signert avtale leveres digitalt i Fronter(IMT3912)
Godkjennes digitalt av AIMTs dekan

Om papirversjon med signatur er ønskelig, må papirversjon leveres til AIMT i tillegg.
Plass for evt sign:
AIMT Dekan/prodekan (signatur): dato

VR terra Agreement

VR terra is an organisation set up to investigate and develop Virtual Reality and terrain models.

This agreement is between
Simon McCallum (20%)
Mårten Nordheim (40%)
Nichlas Severinsen (40%)

The intellectual property developed within the projects this group work on will be owned by the group with the percentage indicated above. Decisions on the direction of the group will be made by majority voting until June 3rd, 2016. On June 3rd 2016 there will be a meeting to decide continued development of the project.

Each individual in the group has the non exclusive right to continue working on the project after the June 3rd meeting. Agreement to form a company and commercially exploit the results of the project will be made in discussion with NTNU Technology Transfer Office.

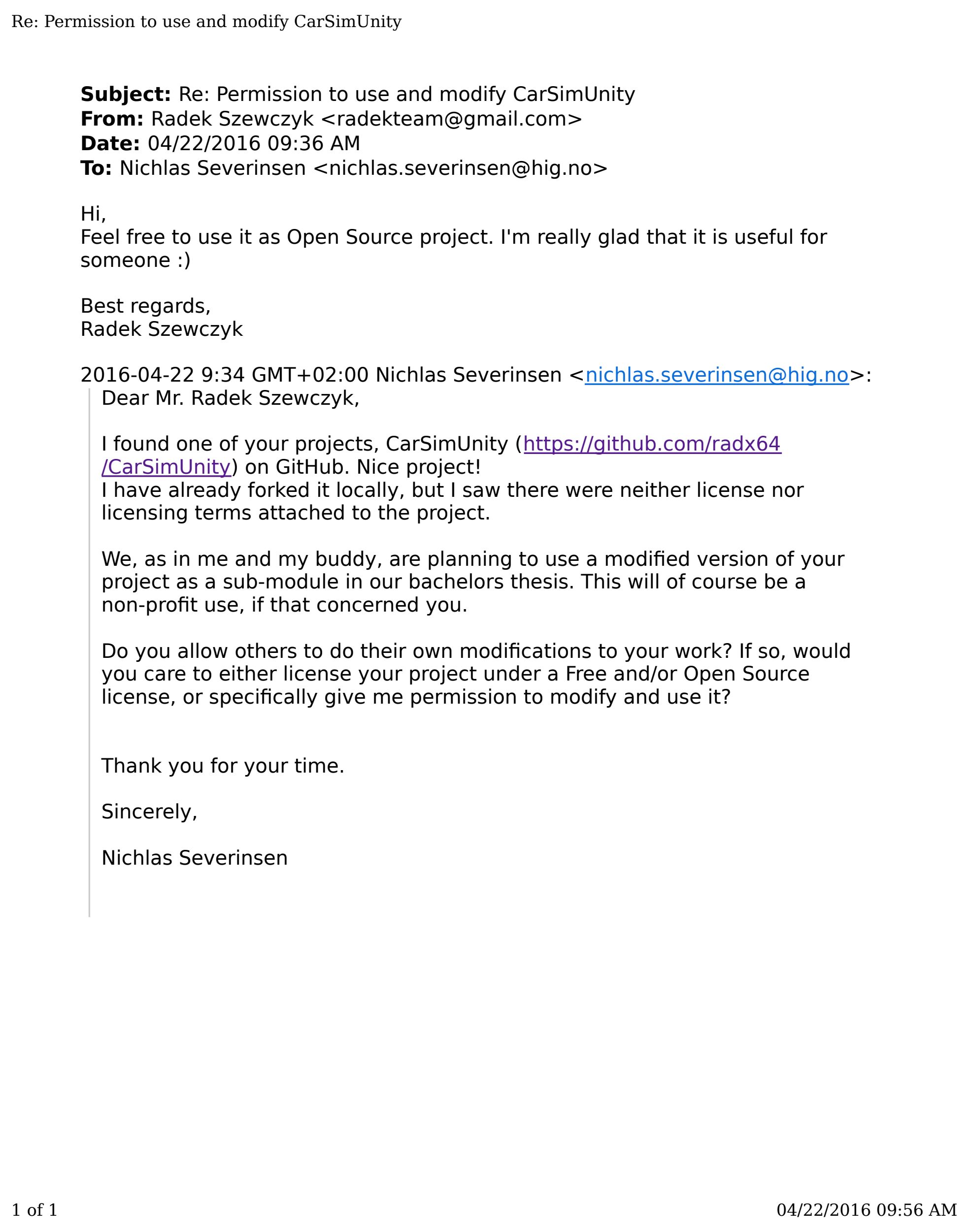
Signed

[illegible]

Mårten Nordheim

[illegible]

I Permission to use CarSimUnity



Re: Permission to use and modify CarSimUnity

Subject: Re: Permission to use and modify CarSimUnity
From: Radek Szewczyk <radekteam@gmail.com>
Date: 04/22/2016 09:36 AM
To: Nichlas Severinsen <nichlas.severinsen@hig.no>

Hi,
Feel free to use it as Open Source project. I'm really glad that it is useful for someone :)

Best regards,
Radek Szewczyk

2016-04-22 9:34 GMT+02:00 Nichlas Severinsen <nichlas.severinsen@hig.no>:

> Dear Mr. Radek Szewczyk,
>
> I found one of your projects, CarSimUnity (https://github.com/radx64/CarSimUnity) on GitHub. Nice project!
> I have already forked it locally, but I saw there were neither license nor licensing terms attached to the project.
>
> We, as in me and my buddy, are planning to use a modified version of your project as a sub-module in our bachelors thesis. This will of course be a non-profit use, if that concerned you.
>
> Do you allow others to do their own modifications to your work? If so, would you care to either license your project under a Free and/or Open Source license, or specifically give me permission to modify and use it?
>
> Thank you for your time.
>
> Sincerely,
>
> Nichlas Severinsen

1 of 1 04/22/2016 09:56 AM

J Toggl Time Tracking Data

Detailed report

2016-01-01 - 2016-05-22
Total 959 h 03 min

Date	Description	Duration	User
01-13	**Working on Project Plan, some latex stuff, some tool research** Bachelor Project	**2:00:00** 08:00-10:00	Nichlas Severinsen
01-13	**Directory structure** Bachelor Project	**0:30:55** 09:29-10:00	Mårten Nordheim
01-13	**Set up Google Test** Bachelor Project	**2:32:13** 10:09-12:41	Mårten Nordheim
01-13	**Working on Project Plan, some latex stuff, some tool research, gantt diagram with planner** Bachelor Project	**1:45:45** 10:17-12:03	Nichlas Severinsen
01-13	**Working on Project Plan, some latex stuff, some tool research, gantt diagram with planner** Bachelor Project	**3:14:15** 12:10-15:24	Nichlas Severinsen
01-13	**Adding required libraries** Bachelor Project	**0:05:01** 13:28-13:33	Mårten Nordheim
01-13	**Adding required libraries** Bachelor Project	**0:00:22** 13:59-13:59	Mårten Nordheim
01-13	**Adding required libraries** Bachelor Project	**0:25:19** 14:03-14:29	Mårten Nordheim
01-14	**WBS, GDD, Project plan** Bachelor Project	**3:51:02** 08:04-11:55	Nichlas Severinsen
01-14	**Research/working on plan** Bachelor Project	**1:44:34** 08:14-09:58	Mårten Nordheim
01-14	**GDD, Project plan, research** Bachelor Project	**2:45:00** 12:13-14:58	Nichlas Severinsen
01-14	**Research/working on plan** Bachelor Project	**2:32:48** 12:25-14:58	Mårten Nordheim
01-15	**Research/working on plan** Bachelor Project	**4:08:32** 07:46-11:55	Mårten Nordheim
01-15	**Project plan, reading other projects; research** Bachelor Project	**2:46:54** 08:00-10:47	Nichlas Severinsen
01-15	**Risks, Project visualization** Bachelor Project	**1:08:01** 10:47-11:55	Nichlas Severinsen
01-15	**Research/working on plan** Bachelor Project	**0:11:16** 12:30-12:42	Mårten Nordheim
01-15	**Research/working on plan** Bachelor Project	**1:29:59** 12:47-14:17	Mårten Nordheim

01-18	**Researching engine/library** Bachelor Project	**4:20:37** 07:46-12:07	Mårten Nordheim
01-18	**Terrain Library; planning, creation, research, OGRE** Bachelor Project	**4:01:37** 08:13-12:14	Nichlas Severinsen
01-18	**Terrain Library; planning, creation, research, OGRE** Bachelor Project	**0:18:21** 12:29-12:47	Nichlas Severinsen
01-18	**Researching engine/library** Bachelor Project	**1:19:59** 12:30-13:50	Mårten Nordheim
01-19	**Engine** Bachelor Project	**3:51:10** 08:08-11:59	Mårten Nordheim
01-19	**Terrain library, tests** Bachelor Project	**3:45:43** 08:19-12:04	Nichlas Severinsen
01-19	**Engine** Bachelor Project	**2:38:29** 12:18-14:57	Mårten Nordheim
01-19	**Terrain library, tests** Bachelor Project	**1:19:06** 12:39-13:58	Nichlas Severinsen
01-19	**Project plan** Bachelor Project	**0:57:42** 13:58-14:56	Nichlas Severinsen
01-19	**Doxygen** Bachelor Project	**0:02:00** 18:49-18:51	Mårten Nordheim
01-20	**Engine** Bachelor Project	**5:15:05** 07:50-13:05	Mårten Nordheim
01-20	**Make what mårten made yesterday, base engine, compile on linux** Bachelor Project	**0:31:41** 08:01-08:33	Nichlas Severinsen
01-20	**study mårtens code** Bachelor Project	**0:25:52** 08:33-08:58	Nichlas Severinsen
01-20	**make terrain library return vertices, tests** Bachelor Project	**2:06:00** 08:59-11:05	Nichlas Severinsen
01-20	**Terrain library!** Bachelor Project	**1:55:07** 11:05-13:00	Nichlas Severinsen
01-20	**Terrain library, circle function** Bachelor Project	**0:06:05** 13:00-13:06	Nichlas Severinsen
01-20	**Engine** Bachelor Project	**1:46:54** 13:11-14:58	Mårten Nordheim
01-20	**Terrain library, circle function** Bachelor Project	**1:15:12** 13:43-14:58	Nichlas Severinsen
01-21	**Engine** Bachelor Project	**2:10:16** 07:47-09:57	Mårten Nordheim
01-21	**Terrain library, circle function; having problems protecting loop** Bachelor Project	**1:42:00** 08:00-09:42	Nichlas Severinsen
01-21	**Terrain library, circle function; debugging double free or corruption** Bachelor Project	**2:28:30** 09:42-12:10	Nichlas Severinsen

01-21	**Engine** Bachelor Project	**0:35:34** 11:26-12:01	Mårten Nordheim
01-21	**Engine** Bachelor Project	**2:52:10** 12:05-14:58	Mårten Nordheim
01-21	**Terrain library, refactoring, pair programming** Bachelor Project	**0:35:02** 12:10-12:45	Nichlas Severinsen
01-21	**Terrain library, refactoring, smoothing function research** Bachelor Project	**1:30:01** 13:11-14:41	Nichlas Severinsen
01-22	**Terrain library, refactoring, smoothing function research** Bachelor Project	**4:00:00** 08:00-12:00	Nichlas Severinsen
01-22	**Engine** Bachelor Project	**4:08:09** 08:27-12:35	Mårten Nordheim
01-22	**Sprint review, retrospective, planning** Bachelor Project	**1:35:00** 12:15-13:50	Nichlas Severinsen
01-22	**Engine** Bachelor Project	**1:09:20** 12:48-13:57	Mårten Nordheim
01-23	**Engine / catching up on lost hours during week** Bachelor Project	**0:45:26** 20:34-21:19	Mårten Nordheim
01-24	**Engine / catching up on lost hours during week** Bachelor Project	**1:19:45** 16:06-17:26	Mårten Nordheim
01-25	**study mårtens code, check that it compiles on gnu/linux etc** Bachelor Project	**2:00:00** 08:00-10:00	Nichlas Severinsen
01-25	**Engine** Bachelor Project	**3:49:19** 08:26-12:16	Mårten Nordheim
01-25	**Set up networking, research, fighting with cmake** Bachelor Project	**4:45:00** 10:00-14:45	Nichlas Severinsen
01-25	**Engine** Bachelor Project	**2:15:29** 12:43-14:58	Mårten Nordheim
01-26	**Finally got ENet to work** Bachelor Project	**3:00:00** 08:00-11:00	Nichlas Severinsen
01-26	**Engine** Bachelor Project	**2:26:44** 08:35-11:02	Mårten Nordheim
01-26	**Engine** Bachelor Project	**1:18:29** 13:41-15:00	Mårten Nordheim
01-27	**Project plan, meeting with Simon, polishing, signing** Bachelor Project	**4:00:00** 08:00-12:00	Nichlas Severinsen
01-27	**Engine** Bachelor Project	**0:08:10** 08:50-08:58	Mårten Nordheim
01-27	**Engine** Bachelor Project	**1:20:37** 11:14-12:35	Mårten Nordheim
01-27	**Networking!** Bachelor Project	**2:45:00** 12:15-15:00	Nichlas Severinsen

01-27	**Engine / refactoring** Bachelor Project	**2:16:14** 12:42-14:58	Mårten Nordheim
01-28	**final Project plan, project agreement, hand in.** Bachelor Project	**0:19:00** 08:00-08:19	Nichlas Severinsen
01-28	**Networking!** Bachelor Project	**3:41:00** 08:19-12:00	Nichlas Severinsen
01-28	**Engine / refactoring** Bachelor Project	**3:26:32** 08:28-11:54	Mårten Nordheim
01-28	**Networking!** Bachelor Project	**2:45:00** 12:15-15:00	Nichlas Severinsen
01-28	**Engine / refactoring** Bachelor Project	**1:52:00** 13:09-15:01	Mårten Nordheim
01-29	**Networking! fighting with protocol buffers, sprint revie/retro/plan at the end** Bachelor Project	**5:06:21** 08:35-13:41	Nichlas Severinsen
01-29	**Engine** Bachelor Project	**3:19:15** 08:35-11:55	Mårten Nordheim
01-29	**Engine / load config file** Bachelor Project	**1:18:44** 12:12-13:31	Mårten Nordheim
02-01	**Networking! fighting with protocol buffers, sprint revie/retro/plan at the end** Bachelor Project	**2:00:00** 08:00-10:00	Nichlas Severinsen
02-01	**Engine / load config file** Bachelor Project	**2:56:23** 08:08-11:04	Mårten Nordheim
02-01	**Lighting, normals, rendering** Bachelor Project	**2:00:00** 10:00-12:00	Nichlas Severinsen
02-01	**Engine / networking** Bachelor Project	**3:55:44** 11:05-15:00	Mårten Nordheim
02-01	**Lighting, normals, rendering** Bachelor Project	**2:30:00** 12:30-15:00	Nichlas Severinsen
02-02	**Engine / networking** Bachelor Project	**4:00:01** 08:02-12:02	Mårten Nordheim
02-02	**Lighting, normals, rendering, researching clicking** Bachelor Project	**6:26:58** 08:21-14:48	Nichlas Severinsen
02-02	**Engine / networking** Bachelor Project	**2:48:25** 12:15-15:04	Mårten Nordheim
02-03	**Networking** Bachelor Project	**6:45:29** 08:11-14:57	Mårten Nordheim
02-03	**Terrain editor UI(rendercomponent + handler)** Bachelor Project	**2:30:00** 09:30-12:00	Nichlas Severinsen
02-03	**Terrain editor UI(rendercomponent + handler), research, opengl, stuff** Bachelor Project	**2:30:00** 12:30-15:00	Nichlas Severinsen
02-04	**Networking** Bachelor Project	**3:42:38** 08:07-11:49	Mårten Nordheim

02-04	Terrain editor UI(rendercomponent + handler), research, opengl, stuff Bachelor Project	2:30:00 09:30-12:00	Nichlas Severinsen
02-04	Terrain editor UI(rendercomponent + handler), research, opengl, stuff Bachelor Project	2:30:00 12:30-15:00	Nichlas Severinsen
02-04	Networking Bachelor Project	2:40:41 12:58-15:39	Mårten Nordheim
02-05	Terrain editor UI(rendercomponent + handler), research, opengl, stuff Bachelor Project	1:00:00 08:00-09:00	Nichlas Severinsen
02-05	Networking Bachelor Project	3:47:44 08:03-11:50	Mårten Nordheim
02-05	Networking Bachelor Project	1:39:40 12:05-13:45	Mårten Nordheim
02-08	Networking Bachelor Project	6:38:27 08:17-14:56	Mårten Nordheim
02-08	Mousepicking + Normals Bachelor Project	6:00:00 09:00-15:00	Nichlas Severinsen
02-09	Networking Bachelor Project	7:13:47 08:03-15:16	Mårten Nordheim
02-09	Mousepicking, Normals, Shader, Lighting Bachelor Project	6:00:00 08:30-14:30	Nichlas Severinsen
02-09	Networking Bachelor Project	0:26:04 15:16-15:43	Mårten Nordheim
02-10	Meeting with simon, some project writing, looked at CI and tried a ton of them Bachelor Project	6:45:00 08:00-14:45	Nichlas Severinsen
02-10	Networking Bachelor Project	8:57:17 08:01-16:59	Mårten Nordheim
02-11	Testing out more 3D engines, trying to get kinect to work Bachelor Project	7:00:00 08:00-15:00	Nichlas Severinsen
02-11	Research / Torque3d Bachelor Project	3:30:46 08:15-11:45	Mårten Nordheim
02-11	Research / Torque3d Bachelor Project	2:00:46 12:58-14:59	Mårten Nordheim
02-12	Irrlicht research, testing, tutorials, and it works! Bachelor Project	2:30:00 08:00-10:30	Nichlas Severinsen
02-12	Torque3d Bachelor Project	5:23:22 08:22-13:45	Mårten Nordheim
02-12	Sprint review, retro, planning Bachelor Project	1:35:00 12:15-13:50	Nichlas Severinsen
02-15	Attempting w/ terrain deformation in irrlicht Bachelor Project	6:33:37 08:20-14:54	Mårten Nordheim
02-15	Irrlicht Bachelor Project	6:15:00 08:30-14:45	Nichlas Severinsen

02-16	**Irrlicht set-up and adding old libraries** Bachelor Project	**6:54:50** 08:02-14:57	Mårten Nordheim
02-16	**Irrlicht** Bachelor Project	**4:45:00** 09:30-14:15	Nichlas Severinsen
02-17	**Irrlicht, base irrlicht class** Bachelor Project	**7:00:00** 08:00-15:00	Nichlas Severinsen
02-17	**Troubleshooting + networking** Bachelor Project	**7:04:29** 08:02-15:07	Mårten Nordheim
02-18	**Irrlicht, GUI** Bachelor Project	**7:00:00** 08:00-15:00	Nichlas Severinsen
02-18	**Networking + Kinect** Bachelor Project	**6:53:25** 08:10-15:04	Mårten Nordheim
02-19	**Irrlicht, GUI** Bachelor Project	**6:00:00** 08:00-14:00	Nichlas Severinsen
02-19	**Kinect** Bachelor Project	**5:33:32** 08:10-13:43	Mårten Nordheim
02-22	**Irrlicht, GUI** Bachelor Project	**6:00:00** 08:00-14:00	Nichlas Severinsen
02-22	**Kinect + setting up linux env** Bachelor Project	**6:43:47** 08:22-15:05	Mårten Nordheim
02-23	**Irrlicht, GUI** Bachelor Project	**6:00:00** 08:00-14:00	Nichlas Severinsen
02-23	**VRui/Kinect/SARndbox source and testing** Bachelor Project	**6:51:13** 08:05-14:56	Mårten Nordheim
02-24	**Kinect** Bachelor Project	**6:45:00** 08:00-14:45	Nichlas Severinsen
02-24	**Looking into strange Kinect video output** Bachelor Project	**6:57:39** 08:05-15:02	Mårten Nordheim
02-25	**Kinect, SARNdbox** Bachelor Project	**6:45:00** 08:00-14:45	Nichlas Severinsen
02-25	**Studying SARndbox code** Bachelor Project	**3:33:33** 08:09-11:43	Mårten Nordheim
02-25	**(no description)** Bachelor Project	**0:05:04** 12:38-12:43	Mårten Nordheim
02-25	**Picked up a desktop and a projector** Bachelor Project	**1:34:43** 14:05-15:40	Mårten Nordheim
02-26	**Kinect, SARNdbox** Bachelor Project	**6:00:00** 08:00-14:00	Nichlas Severinsen
02-26	**Setting up the desktop and end-of-sprint activities** Bachelor Project	**5:43:51** 08:05-13:49	Mårten Nordheim
02-29	**Remove RAID+Windows from computer and install/setup GNU/Linux Mint** Bachelor Project	**1:30:00** 08:00-09:30	Nichlas Severinsen

02-29	**SARndbox studying, updating terrain from network packets** Bachelor Project	**6:12:56** 08:10-14:23	Mårten Nordheim
02-29	**Trying to color heightmap vertexes based on height** Bachelor Project	**4:45:00** 09:30-14:15	Nichlas Severinsen
03-01	**Made some progress on colored heigthmap by following meshtutorial 23** Bachelor Project	**7:30:00** 08:00-15:30	Nichlas Severinsen
03-01	**Updating terrain from network packets** Bachelor Project	**7:32:34** 08:03-15:36	Mårten Nordheim
03-02	**Updating terrain from network packets** Bachelor Project	**2:57:25** 07:35-10:33	Mårten Nordheim
03-02	**Refactoring to get heightmap color to work** Bachelor Project	**2:30:00** 08:00-10:30	Nichlas Severinsen
03-02	**Quick meeting with simon** Bachelor Project	**0:10:00** 10:45-10:55	Nichlas Severinsen
03-02	**Hamar** Bachelor Project	**4:00:01** 11:00-15:00	Mårten Nordheim
03-02	**@ Hamar Game Collective** Bachelor Project	**2:45:00** 12:00-14:45	Nichlas Severinsen
03-03	**Updating terrain from network packets** Bachelor Project	**4:54:08** 07:52-12:46	Mårten Nordheim
03-03	**Finishing work on terrain mesh, ran into segfault problems** Bachelor Project	**1:50:00** 08:00-09:50	Nichlas Severinsen
03-03	**Debugging terrainmesh segfault, decided trying to make a terrainheightmap** Bachelor Project	**2:00:00** 11:00-13:00	Nichlas Severinsen
03-03	**Looked into reading heightmap from kinect somehow, ended up with depth image** Bachelor Project	**2:30:00** 13:10-15:40	Nichlas Severinsen
03-03	**Updating terrain from network packets** Bachelor Project	**1:45:20** 14:06-15:51	Mårten Nordheim
03-04	**Trying to read depthbuffer from kinect, pluss some API** Bachelor Project	**5:30:00** 08:00-13:30	Nichlas Severinsen
03-04	**Updating terrain from network packets** Bachelor Project	**5:51:35** 08:13-14:05	Mårten Nordheim
03-07	**Kinect hacking with Mårten :), pear!** Bachelor Project	**6:45:00** 08:00-14:45	Nichlas Severinsen
03-07	**Updating terrain from network packets, working on kinect** Bachelor Project	**7:00:42** 08:07-15:07	Mårten Nordheim
03-08	**Research, refactoring kinect** Bachelor Project	**2:45:00** 08:00-10:45	Nichlas Severinsen
03-08	**Researched easy VR solutions** Bachelor Project	**6:42:04** 08:14-14:56	Mårten Nordheim

03-08	**Research, refactoring kinect, some report writing** Bachelor Project	**2:45:00** 11:45-14:30	Nichlas Severinsen
03-09	**Getting kinect data into the game and using it as heightmap as an example** Bachelor Project	**3:40:00** 08:00-11:40	Nichlas Severinsen
03-09	**Researched VR** Bachelor Project	**3:42:44** 08:01-11:44	Mårten Nordheim
03-09	**Lynkurs** Bachelor Project	**1:18:48** 11:44-13:03	Mårten Nordheim
03-09	**Optimizing updating irrlicht's terrain** Bachelor Project	**1:55:25** 13:03-14:58	Mårten Nordheim
03-09	**Getting kinect data into the game and using it as heightmap as an example** Bachelor Project	**1:15:00** 13:30-14:45	Nichlas Severinsen
03-10	**Struggling with properly render kinect data** Bachelor Project	**6:45:00** 08:00-14:45	Nichlas Severinsen
03-10	**Working on issue regarding updating irrlicht's terrain from network** Bachelor Project	**4:37:26** 08:08-12:46	Mårten Nordheim
03-10	**Working on issue regarding updating irrlicht's terrain from network** Bachelor Project	**1:09:23** 13:53-15:02	Mårten Nordheim
03-11	**Fixed issue regarding updating irrlicht's terrain from network** Bachelor Project	**8:18:43** 08:00-16:19	Mårten Nordheim
03-14	**Small refactoring + Started on a shader for the terrain** Bachelor Project	**6:51:41** 08:12-15:04	Mårten Nordheim
03-15	**Working on shader** Bachelor Project	**4:04:06** 11:48-15:52	Mårten Nordheim
03-16	**Working on implementing a TCP server** Bachelor Project	**3:24:51** 12:17-15:42	Mårten Nordheim
03-16	**Working on implementing a TCP server** Bachelor Project	**0:27:02** 16:02-16:29	Mårten Nordheim
03-17	**Working on implementing a TCP server + JSON-based packets** Bachelor Project	**5:57:36** 10:08-16:05	Mårten Nordheim
03-18	**Working on implementing a TCP server + Rewriting some of Terrain as Copy-On-Write** Bachelor Project	**4:41:56** 11:15-15:56	Mårten Nordheim
03-19	**Tested and fixed COW stuff in Terrain** Bachelor Project	**1:00:00** 13:18-14:18	Mårten Nordheim
03-21	**Work on shader** Bachelor Project	**1:00:00** 13:00-14:00	Mårten Nordheim
03-29	**(no description)** Bachelor Project	**2:30:00** 12:15-14:45	Mårten Nordheim
03-30	**First day after hospital+easter; refactoring, projector program, smaller changes** Bachelor Project	**4:00:00** 08:00-12:00	Nichlas Severinsen

03-30	**Refactoring** Bachelor Project	**6:59:44** 08:00-15:00	Mårten Nordheim
03-30	**First day after hospital+easter; refactoring, projector program, smaller changes** Bachelor Project	**2:15:00** 12:30-14:45	Nichlas Severinsen
03-30	**Fixed weird issue I had** Bachelor Project	**0:20:01** 20:11-20:31	Mårten Nordheim
03-31	**projector, merged in mårtens base refactor** Bachelor Project	**4:00:00** 08:00-12:00	Nichlas Severinsen
03-31	**Optimized copying in an entire vector into the heightmap** Bachelor Project	**6:44:44** 08:10-14:54	Mårten Nordheim
03-31	**projector, struggling with segfault** Bachelor Project	**2:15:00** 12:30-14:45	Nichlas Severinsen
04-01	**setting up project on borrowed computer for projection+server** Bachelor Project	**4:00:00** 08:00-12:00	Nichlas Severinsen
04-01	**Set up the desktop to run the projector and tweaked the shader to try and make it look ok** Bachelor Project	**7:00:01** 08:51-15:51	Mårten Nordheim
04-01	**setting up project on borrowed computer for projection+server** Bachelor Project	**2:15:00** 12:30-14:45	Nichlas Severinsen
04-04	**Filling in and editing report** Bachelor Project	**4:00:00** 08:00-12:00	Nichlas Severinsen
04-04	**Playing around with shader + fixed small race condition** Bachelor Project	**6:52:10** 08:05-14:57	Mårten Nordheim
04-04	**Filling in and editing report** Bachelor Project	**2:30:00** 12:30-15:00	Nichlas Severinsen
04-05	**Look at the worklog** Bachelor Project	**7:09:50** 07:50-15:00	Mårten Nordheim
04-05	**Filling in and editing report** Bachelor Project	**1:15:00** 08:00-09:15	Nichlas Severinsen
04-05	**Filling in and editing report** Bachelor Project	**2:15:00** 09:45-12:00	Nichlas Severinsen
04-05	**Filling in and editing report** Bachelor Project	**2:25:00** 12:30-14:55	Nichlas Severinsen
04-06	**Filling in and editing report** Bachelor Project	**4:00:00** 08:00-12:00	Nichlas Severinsen
04-06	**Improving terrain transmission** Bachelor Project	**6:54:58** 08:05-15:00	Mårten Nordheim
04-06	**Filling in and editing report** Bachelor Project	**2:15:00** 12:30-14:45	Nichlas Severinsen
04-07	**Wrote tests + stuck with a rendering issue** Bachelor Project	**7:03:02** 07:58-15:01	Mårten Nordheim

04-07	**Filling in and editing report** Bachelor Project	**4:00:00** 08:00-12:00	Nichlas Severinsen
04-07	**Filling in and editing report** Bachelor Project	**2:15:00** 12:30-14:45	Nichlas Severinsen
04-08	**Banging my head against the rendering issue** Bachelor Project	**2:05:15** 07:50-09:55	Mårten Nordheim
04-08	**Game branch. merge in kinect and refactor to use base** Bachelor Project	**4:15:00** 08:00-12:15	Nichlas Severinsen
04-08	**Manually fixing the rendering issue** Bachelor Project	**2:40:34** 11:23-14:03	Mårten Nordheim
04-08	**Game branch. merge in kinect and refactor to use base** Bachelor Project	**1:00:00** 12:15-13:15	Nichlas Severinsen
04-08	**Sprint review, retro, planning** Bachelor Project	**0:30:00** 13:15-13:45	Nichlas Severinsen
04-11	**researching open source VR games** Bachelor Project	**4:00:00** 08:00-12:00	Nichlas Severinsen
04-11	**planning new tcp features** Bachelor Project	**6:44:41** 08:15-15:00	Mårten Nordheim
04-11	**researching open source VR games** Bachelor Project	**2:15:00** 12:30-14:45	Nichlas Severinsen
04-12	**UNITY** Bachelor Project	**2:00:00** 08:00-10:00	Nichlas Severinsen
04-12	**implementing new tcp features** Bachelor Project	**7:03:43** 08:00-15:03	Mårten Nordheim
04-12	**Searching for open source unity games** Bachelor Project	**2:00:00** 10:00-12:00	Nichlas Severinsen
04-12	**Searching for open source unity games** Bachelor Project	**0:45:00** 12:15-13:00	Nichlas Severinsen
04-12	**vr-game, fork of CarSimUnity** Bachelor Project	**1:45:00** 13:00-14:45	Nichlas Severinsen
04-13	**UNITY** Bachelor Project	**4:00:00** 08:00-12:00	Nichlas Severinsen
04-13	**implementing new tcp features** Bachelor Project	**6:58:00** 08:02-15:00	Mårten Nordheim
04-13	**UNITY** Bachelor Project	**2:40:00** 12:15-14:55	Nichlas Severinsen
04-14	**implementing new tcp features** Bachelor Project	**6:42:10** 08:18-15:00	Mårten Nordheim
04-15	**implementing new tcp features** Bachelor Project	**5:35:47** 07:58-13:34	Mårten Nordheim
04-18	**implementing new tcp features** Bachelor Project	**7:00:00** 08:00-15:00	Mårten Nordheim

04-19	**Report** Bachelor Project	**1:46:04**	Mårten Nordheim
04-19	**møte: vitensenteret** Bachelor Project	**0:53:44** 09:47-10:41	Mårten Nordheim
04-19	**Report** Bachelor Project	**4:20:00** 10:40-15:00	Nichlas Severinsen
04-19	**C# tcp client to test the tcp impl.** Bachelor Project	**4:19:00** 10:41-15:00	Mårten Nordheim
04-20	**Report** Bachelor Project	**4:00:00** 08:00-12:00	Nichlas Severinsen
04-20	**draw marker based on tcp input** Bachelor Project	**6:59:29** 08:00-15:00	Mårten Nordheim
04-20	**Report** Bachelor Project	**2:45:00** 12:15-15:00	Nichlas Severinsen
04-21	**Report** Bachelor Project	**7:00:00** 08:00-15:00	Nichlas Severinsen
04-21	**report** Bachelor Project	**6:55:00** 08:05-15:00	Mårten Nordheim
04-22	**Report, Deployment, CarSimUnity Permission** Bachelor Project	**6:00:00** 08:00-14:00	Nichlas Severinsen
04-22	**report** Bachelor Project	**5:34:35** 08:05-13:40	Mårten Nordheim
04-25	**Merging together various branches and correcting mistakes of auto-merge + moved from mustad to the gamelab** Bachelor Project	**7:42:37** 07:50-15:33	Mårten Nordheim
04-25	**sandbox, report, etc** Bachelor Project	**7:00:00** 08:00-15:00	Nichlas Severinsen
04-26	**terrain mapping** Bachelor Project	**4:00:00** 08:00-12:00	Nichlas Severinsen
04-26	**remove padding + report + pair programming** Bachelor Project	**6:33:31** 08:21-14:54	Mårten Nordheim
04-26	**terrain mapping** Bachelor Project	**2:30:00** 12:15-14:45	Nichlas Severinsen
04-27	**report + texture mapping** Bachelor Project	**6:37:13** 08:15-14:53	Mårten Nordheim
04-27	**terrain mapping** Bachelor Project	**2:00:00** 10:00-12:00	Nichlas Severinsen
04-27	**terrain mapping** Bachelor Project	**2:45:00** 12:15-15:00	Nichlas Severinsen
04-27	**texture mapping** Bachelor Project	**0:20:00** 17:12-17:32	Mårten Nordheim
04-28	**terrain mapping** Bachelor Project	**7:00:00** 08:00-15:00	Nichlas Severinsen

04-28	**texture mapping** Bachelor Project	**6:43:16** 08:04-14:48	Mårten Nordheim
04-29	**texture should fill the screen** Bachelor Project	**6:23:40** 08:01-14:24	Mårten Nordheim
05-01	**restored enet server/client-functionality** Bachelor Project	**0:30:00** 15:13-15:43	Mårten Nordheim
05-02	**report** Bachelor Project	**7:00:00** 08:00-15:00	Nichlas Severinsen
05-02	**Refactoring Base** Bachelor Project	**6:31:38** 08:15-14:46	Mårten Nordheim
05-03	**report** Bachelor Project	**7:00:00** 08:00-15:00	Nichlas Severinsen
05-03	**Finishing refactoring** Bachelor Project	**6:50:18** 08:00-14:50	Mårten Nordheim
05-04	**extracting terrain selection, shading with height** Bachelor Project	**6:48:20** 07:59-14:48	Mårten Nordheim
05-04	**report** Bachelor Project	**7:00:00** 08:00-15:00	Nichlas Severinsen
05-05	**rEport** Bachelor Project	**5:10:56** 10:21-15:32	Mårten Nordheim
05-05	**Report** Bachelor Project	**2:59:34** 15:52-18:51	Mårten Nordheim
05-06	**Report** Bachelor Project	**5:43:35** 08:06-13:50	Mårten Nordheim
05-07	**report** Bachelor Project	**6:00:00** 08:00-14:00	Nichlas Severinsen
05-07	**Refactoring various elements** Bachelor Project	**1:50:00** 15:15-17:05	Mårten Nordheim
05-09	**Averaging frames** Bachelor Project	**3:58:17** 07:51-11:50	Mårten Nordheim
05-09	**report** Bachelor Project	**7:00:00** 08:00-15:00	Nichlas Severinsen
05-09	**Report** Bachelor Project	**2:49:34** 11:51-14:40	Mårten Nordheim
05-10	**report** Bachelor Project	**7:00:00** 08:00-15:00	Nichlas Severinsen
05-10	**VR testing** Bachelor Project	**2:29:36** 08:13-10:42	Mårten Nordheim
05-10	**VR testing** Bachelor Project	**1:44:25** 10:43-12:27	Mårten Nordheim
05-10	**extracting terrain selection** Bachelor Project	**2:12:51** 12:29-14:42	Mårten Nordheim
05-10	**extracting terrain selection** Bachelor Project	**2:20:49** 19:38-21:59	Mårten Nordheim

05-10	**extracting terrain selection** Bachelor Project	**0:28:30** 22:24-22:52	Mårten Nordheim
05-11	**report** Bachelor Project	**7:00:00** 08:00-15:00	Nichlas Severinsen
05-11	**extracting terrain selection (compile on linux)** Bachelor Project	**2:50:25** 08:00-10:50	Mårten Nordheim
05-11	**Report** Bachelor Project	**3:58:28** 10:51-14:50	Mårten Nordheim
05-12	**report** Bachelor Project	**7:00:00** 08:00-15:00	Nichlas Severinsen
05-12	**Report** Bachelor Project	**5:23:41** 08:06-13:30	Mårten Nordheim
05-12	**Report** Bachelor Project	**0:45:09** 14:05-14:50	Mårten Nordheim
05-13	**Report** Bachelor Project	**7:18:19** 07:55-15:13	Mårten Nordheim
05-13	**report** Bachelor Project	**7:00:00** 08:00-15:00	Nichlas Severinsen
05-14	**Report** Bachelor Project	**3:20:54** 14:30-17:51	Mårten Nordheim
05-14	**Report** Bachelor Project	**0:27:23** 19:36-20:04	Mårten Nordheim
05-15	**Rereading report and making changes/checking details** Bachelor Project	**3:40:00** 14:15-17:55	Mårten Nordheim

K Professional Programming

This appendix is written as a requirement for the professional programming course which we were taking in parallel to the bachelor project.

K.1 Refactoring

In this section we will go through some of the refactoring we went through during the project. The subsection will signify who did what.

K.1.1 Mårten

We were writing all the functionality we needed into Base for a while, but it became really messy, and hard to maintain and navigate. So I created two subclasses and moved all the relevant content into their relevant class. The lines in and of themselves are not super-interesting to read, but there was a lot of code that got moved. If we had known from the start that Base would snowball into the mess it became we would likely have split it up a lot sooner.
https://bitbucket.org/vrterra/irrlicht/commits/e696558628fedfac77647e7ed1fe02bd8a029f4f

In updating the terrain we initially only had the possibility to update one value at a time, which was slow to process because of the copy-on-write structure which the terrain class had. I then wrote a function which would loop over a buffer and copy in all of them, which fits the copy-on-write structure a lot better as it did not create numerous copies, only 1 was required.

We went from calling this in a loop:
https://bitbucket.org/vrterra/irrlicht/src/961797ce41a5f40cd3551326566a20fdb0ef020b/src/Terrain/Terrain.cpp?fileviewer=file-view-default#Terrain.cpp-125

To calling this once:
https://bitbucket.org/vrterra/irrlicht/src/961797ce41a5f40cd3551326566a20fdb0ef020b/src/Terrain/Terrain.cpp?fileviewer=file-view-default#Terrain.cpp-181

The first function was eventually removed as it was no longer needed.

K.1.2 Nichlas

After starting to use Irrlicht we suddenly had many different main source files for each "project"; one for projector, one for game, and one for editor, etc. Many of the things being done could be put together in a "Base" class so we would not have to have duplicated code all over, so I did just that, making most – if not all – projects depend on this Base class. It later started getting cluttered so Mårten further expanded and fixed it as he wrote above.

K.2 Tools

In the beginning of the project we looked at different tools we could use, as replacements for things but also to explore. This section includes descriptions of these tools, what we thought of them and optionally review, opinions, or experiences.

K.2.1 Agile Tools

This section include tools we looked at for agile software development.

Trello

Does not integrate with Bitbucket without using a 3rd-party service.

TheBugGenie

The version we tested could not integrate with git. And our account would become inaccessible if the password contained non-alphanumeric characters (which our passwords usually do.).

Taiga

Free and open source. It was previously unknown to us so we were curious how it was. First impression was that it had what we needed; semantic commits, webhooks for integration, issue tracking, wiki. However, it works a little poorly and the issue tracking is restricted. No way to add issues to the backlog or as a sub-item to a user story. The semantic commit syntax (as talked about earlier in the project management section of Process), e.g. "TG-REF6465 #Closed", seems arbitrary and is hard to remember.

K.2.2 Continuous Integration

We looked at setting up a Continuous Integration service for our project so both of us could be sure that their newly submitted code would run on the other's computer.

TeamCity

Having no previous experience with TeamCity we wanted to try it out and see how it was to use, so Nichlas tried setting it up on his VPS. However we could not get it to pull from the git repo as it was unable to establish a connection and we could not figure out why.

Jenkins

Mårten previously set up a Jenkins instance for the Game Programming project during the previous semester and had it build newly pushed code on Linux and Windows and it worked great. However, he used his own computer as the host server during that semester and could not do it during this semester. We were able to set up the instance and had it working but the server we were attempting to use was not powerful enough as we needed to host a virtualized instance of Windows, and we had to drop the idea.

L Git Repos

During the project we created multiple repositories, this chapter includes links to each and a short description of it. The bitbucket team url is https://bitbucket.org/vrterra/

L.1 Irrlicht

This is the repository which contains the work we did after switching to Irrlicht. This is the main repository and contains the majority of our code. All the code is currently merged in to the master branch, except for a feature we did not get around to finishing at the time of writing.
https://bitbucket.org/vrterra/irrlicht

L.2 "Application"

This was our original repository, where we started working. It has some OpenGL code and a few classes. Do note that most of the work there is contained in branches different from master as we did not get as far as to start merging in any significant work. The repository was named "Application" as this was where we worked on the code while the only other repository at the time was "Report" where we worked on the report.
https://bitbucket.org/vrterra/application

L.3 c-sharp_tcp_test

This is a repository Mårten made to store the simple C# test application he used to test the TCP server. It was originally not thought to be shared, nor was it worked on by multiple people, so it does not follow feature branching strategy.
https://bitbucket.org/vrterra/c-sharp_tcp_test

L.4 Unity FPS VR Demo

This is the repository created to contain the first person VR demo which was created to show off how the terrain would look after being extracted from the server and then applied to a terrain in VR.
https://bitbucket.org/vrterra/unity-fps-vr-demo