NTNU

Norwegian University of
Science and Technology

Report

IMT4888 Specialisation in Game Technology (2017 Fall)

Magic Sand Buggy

Gjøvik, 16/11/2017

Jan Tiemann

1 Introduction

This is a report about my project work in the course IMT4888 Specialisation in Game Technology. I have never written a report like this in my study career and I hope I meet the expectations.

The aim of this project was to build upon the results of Skotvold, Somby, M´Hamed, Lecoq and extend it to a full multi-player Virtual Reality(VR) / real sandbox game. The main vision was a cooperative racing game with a changing racetrack. One player is in VR driving a car around in a landscape. The landscape is based of the height profile of a real world sandbox. Another player or a group of players are at the sandbox and manipulate the sand. Changes in the real world sand are reflected by the in-game landscape at runtime. The position of the car and other race track elements along with the height profile are projected onto the sand. The aim of the game is to allow the driver to perform specific maneuvers. For example, if the current objective is to jump a long distance. The sandbox players have to form a hill with a long enough runway in-front of it to allow the driving player jump with enough speed off the hill.

Because of lacking manpower and changes in the underlying technology only a proof-of-concept could be realized.

1.1 Sandbox Applications

The foundation for this and the previous project was the Augmented Reality (AR) Sandbox created by researchers at UC Davis[1]. The later used Software *Magic Sand*, which is a partial port of the *AR Sandbox* to the *openframeworks* framework, was mainly written by Thomas Wolf[2]. Both use the same setup. A Microsoft *Kinect* and a projector are installed over a sandbox and face the sand surface. The depth sensors of the *Kinect* allow the application to scan the height of the surface. Afterwards, the projector is used to display this height map back on the sand. Extensive calibration between *Kinect* depth image and back projection is needed to align the projection with the sand surface.

The internal code structure of the *AR Sandbox* project was very opaque and made creating extensions difficult. Individual lines of code were commented but the architecture was not clear and the application was not consistently modularized. It felt like a further developed prototype. At 1/3 of the project time the alternative *Magic Sand* was found and a switch was made. *Magic Sand* was very well modularized. In addition to that, it was build on top of the *openframeworks* framework which made OS-transparent development and preparing the build environment easier.

[1] `https://arsandbox.ucdavis.edu`
[2] `https://github.com/thomwolf/Magic-Sand`

1.2 Previous Work

The previous work was created for the fulfillment of a bachelor level course's requirement. It consisted of two modules. One server which was an extension to the *AR Sandbox* and a demo client application which was build with the Unity Engine. The server extracted the height map from the *AR Sandbox* and send it upon a request from the client over an TCP connection. The client rendered the height map by sculpting a plane accordingly. The transaction used a simple protocol which employed packets with a short header and a payload tail. To serialize the height map, the float values were truncated to three digits, converted to strings using the actual number chars and then send as ASCII chars over the network. Additionally the client employed meshes without updated colliders to present the height data. However, those colliders are essential if one wants to drive with something on it. The most important contribution to the current project was to locate where it was possible to extract the height data from *AR Sandbox*. This had to be a quite hard task since the quality of the code structure was lacking, as mentioned before.

When the setup was switched from *AR Sandbox* to *Magic Sand*, I decided to abandon the previous project and write my own implementation. I had to do that for the unity party anyway and it would have been more work to understand the existing server and adapt it to *Magic Sand* that to write it anew. Furthermore *openframeworks* offered an TCP server implementation so half of the server was already done. I kept the general protocol structure but more on that in chapter 2.3.

2 Method

The project had the same structure as the previous one. It was split into client and server. The server was an extension to the sandbox software with the main task of extracting and sending the height data. In addition to that, the server had to receive and render the position of the car back onto the sand in this project. The client had to request height data, sculpt terrain according to it, take user inputs, run a racing game and send the position of the car back to the server.

2.1 Server

As mentioned before was the server an extra module of the sandbox application. It fit into the structure just like some of the games which were already implemented. It used the offered facilities of the *Magic Sand* software without requiring any changes to the core functionality of the application. As the base software was written in C++ the server was written in C++, too.

The height data was extracted from a filtered image of the *Kinect* depth sensors. The filtering was done by *Magic Sand*. After some testing it was obvious that the data was wrong by a non constant offset. Besides the height height image, the used facility offered a base plane equation. If this base plane was added to the data the offset vanished and the rendered surface resembled the actual sand surface. Sadly, it was not clear in which range the values of the height map and base plane resided in. But I guessed and scaled the baseplane height by half of its offset and

the results looked very accurate. (see Figure 1)

The projection of the car on the sand surface was done alike to the drawing of fishes and rabbits of one of the pre-implemented games. Thanks to the good structure of *Magic Sand* it was easy to learn how it was done and easy to adapt to my use. Currently, the player's car is represented by a simple dot. But I am very confident that this can be easily extended to a vector drawing or even sprite using *openframeworks* capabilities.

In addition to the actual server a dummy server was build. It is basically a standalone version of the actual server and serves a simple *sin* wave patter which changes over time. First it was used to test the, later explained, protocol. After the transaction were stable, it allowed me to continue the development of the client independent of the sandbox. Another reason for the relatively high maturity of this dummy application was that the idea of a sandbox game jam was considered. At such an event, not every team would have constant access to the sandbox but it still needs something to test their code against.

TODO: picture of dummy server out put

TODO: picture of offset error

2.2 Client

The client is written in c++ and uses the *Unity* engine. Initially it was planed to use the *Unreal* engine. However, it was not possible to modify terrain at runtime with it. *Unity*, on the other hand, implemented this feature may versions ago.

*Unreal*s example vehicle implementation was more usable for this scenario since it was a sand buggy and served as inspiration for the project name. But because of the previous mentioned reason *Unity* had to be used and with it its example vehicle implementation. Unfortunately, this was a street car with skid mark animations and screeching sounds. Consequently, the driving performance on the bumpy sand surface was poor. A lift of the car's body and colliders along with an increase in the range of the wheel's suspension improved that performance to acceptable levels. (also see Figure 2) There was not enough time/manpower to build proper car physics.

To improve the driving experience further the sand terrain was smoothed by a simple median filter. In the same step the region around the car was protected from change. This was done to avoid having an update move the terrain which was under the car above it and causing the car to fall through the map.

Other filters which could be turned on allowed to reduce heights which were at the maximum

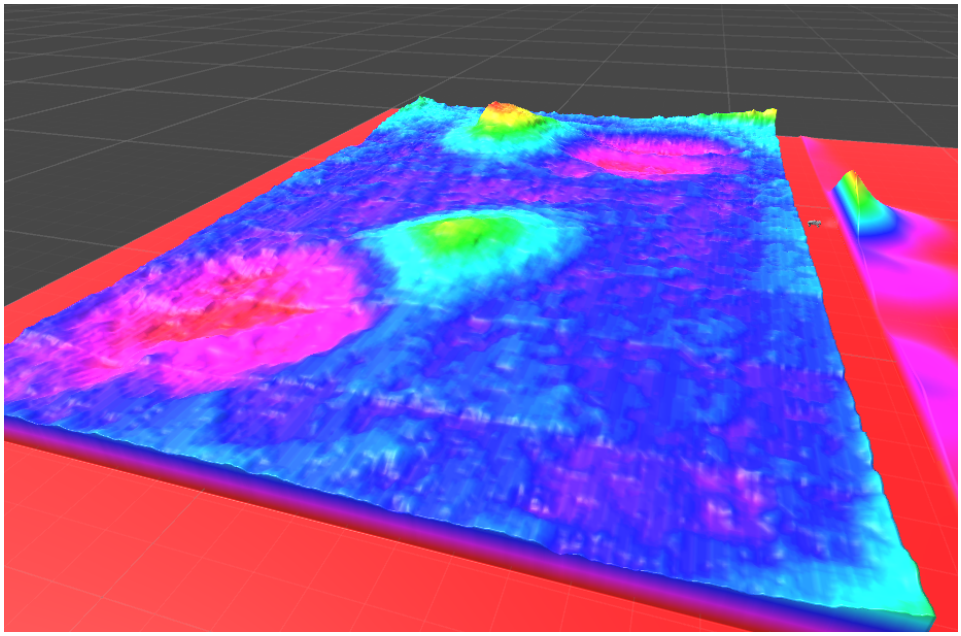

Figure 1: Rendered height map on the client using a height color shader

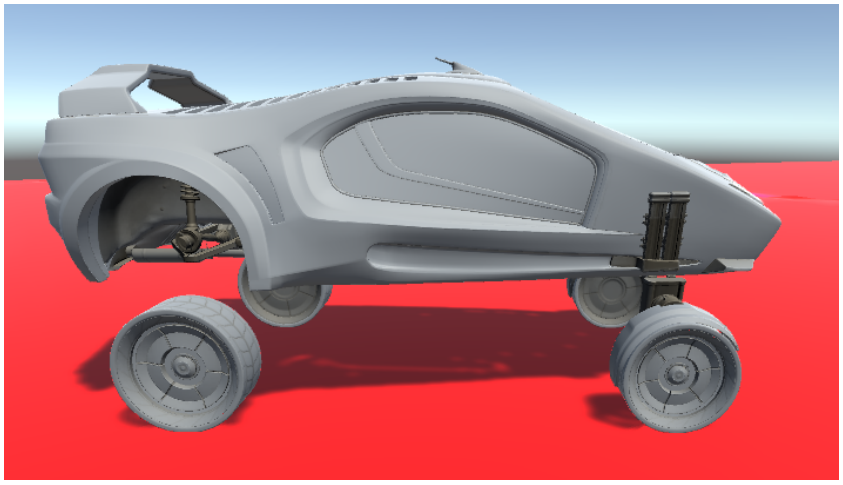

Figure 2: *Unity*'s demo car adjusted for desert use. The vertical lines in the terrain are an artifact of the sandbox application, possibly cause by the too low resolution of the *Kinnect*.

level and scale the height map to the full range. In the early phases of the project the sand surface was not as pronounced as desired. This was especially true since the base plane offset error (mentioned in the previous chapter) was not discovered. To counter this, I introduced a filter that measured the highest and lowest point in the height map and scaled the map in such a way that those points resided at the highest and lowest point possible of the terrain object. Initially, this worked not as desired since everything that was out of *Magic Sand*'s region of interest was set to the highest value. Reducing those values allowed the rescaling to achieve the desired effect. However, caution needed to applied since setting those values to zero simply would have turned the problem on its head.

After the offset error was discovered and fixed, those filters were less needed. But still they help to produce more pronounced looking landscapes. Later it turned out that they are unsuited for use in an actual game scenario. During the game the height of the lowest and highest points change constantly and, even worse, sometimes the body parts of the sandbox players are recognized as high height parts of the sand surface thereby the scaling is constantly changing and the rendered surface feels very unstable for the driving player.

TODO: pictures of (un)scaled/nulled

For a proof-of-concept back-projection which will be later discussed in chapter 3.1 a height color shader hat to be implemented. The height of the terrain was used as hue value of a color in the HSL color model. This color was then translated to RGB and used to paint the corresponding point on the terrain texture. An example can be seen in figure 1.

To test the game in VR, the VR assets were downloaded from the *Unity Asset Store* and put into the game. They worked without an issue and allowed to take a first glimpse of how a finished game may look.

2.3 Protocol

too fast -> people are visible in the sand

using two ports was mistake but no time to fix, maybe not

2.4 Repositories

All modules used *git* as version control. The servers are *openframeorks* apps and need to be put in the corresponding folder to work correctly.

- The *git* repository of the server can be found at:
  `https://github.com/moreApi/Magic-Sand-Buggy`

- The *git* repository of the dummy server can be found at:
  `https://github.com/moreApi/OfServerTest`
- The *git* repository of the client can be found at:
  `https://github.com/moreApi/MagicSandBuggyUnity`

3 Discussion

3.1 Back-Projection

streaming unity cam not doable in time two camera approach - single client different textures for terrain -> different render runs streaming camera out of unity to streaming service is hard and not openly done before two client solution the whole game was build in single player not enouth experience to split it in short time

3.2 Improvements

Was only a single person
buggy graphic, sprite or drawing instead of dot
smoothing between transitons buggy/sand only do (big) changes of terrain when not in view
pick ups
car physics
raceing game stuff

3.3 Ethical Concerns

vs. natural interaction (traditional sandbox), tech/advancemend allways good?
racial body issues, handicaped people
more and more technologial playground ->higher entrance borders - body, monetary issues -> more exclusive

4 Conclusion

will continue, at least for feature jam fun project will ask pb museum if they want sandbox