Project Gaia - Final Report

by

R. Bholanath (4112083), E. van der Hoeven (4109422) and K. de Quillettes (4077482)

Contact Gemeente Delft:
Herman de Haan

Supervisors:
Elmar Eisemann, Rafael Bidarra, Hans-Gerhard Gross and Martha Larson

Faculty of Electrical Engineering, Mathematics and Computer Science

July 2013
A touch table is a perfect way of collaborating within a group. It allows multiple group members to interact with an application at the same time. This in turn allows every member of the group to directly participate in the team process rather than having to send their ideas up the chain of command. An area where such an application might come in useful is in the managing of water flows in a city. This is especially important in Dutch cities, which often lie beneath the sea level. If multiple team members could simultaneously see where there were problems with the water flows in the city and if every team member could present their solutions easily and intuitively, this could be very beneficial.

For the bachelor project, such an application has been developed by the team. The application shows a model of the city of Delft which is accurate with respects to the actual relative sizes of terrain and buildings. The user can then spawn water at any place on the map and observe how the water flows through the city. But the user can do more than observing, he or she can also change the elevation of the terrain. This will allow a user to see how this change affects the flow of water.

The agile programming method scrum was used in the development of the program. This gave the team the flexibility they needed to successfully develop the program. Every team member was assigned a specific role and a planning for every scrum sprint was conceived to make sure that the project would go smoothly. Various tools were also used to this end, these tools handled some of the more mundane aspects of the development or made some hard parts a little easier.

Even though some things did not go as expected during the development, the program as it is at this moment in time fulfills all the goals that were set up beforehand. This means a working prototype is able to run on the touch table of the Gemeente Delft and that every visitor of the i-Lab will soon be able to try out the application.
Preface

This report is the final report for the TI3800 Bachelorproject course at the Delft University of Technology in Delft, the Netherlands. It concludes almost eleven weeks of orientation and development by a team of three students. The assignment for this project was issued by the Gemeente Delft, also located in Delft, the Netherlands and work was primarily done at the faculty of Electrical Engineering, Mathematics and Computer Science of the university of Delft.

First and foremost, we would like to thank our supervisors Elmar Eisemann, Rafael Bidarra and Tim Tutenel. They were all a tremendous help, both in the orientation phase as well as in the actual development phase. We would also like to thank Martha Larson and Hans-Gerhard Gross for coordinating the entire bachelor project. Finally, we would like to thank Herman de Haan, our contact at the Gemeente Delft, who was very helpful and prompt.
Contents

Abstract i

Preface ii

Abbreviations iv

1 Introduction 1
   1.1 Problem Description 1
   1.2 Proposed Solution 2
   1.3 Outline 2

2 Assignment & Requirements 3
   2.1 Assignment Description and Proposed System 3
   2.2 Global Requirements 4
   2.3 List of Requirements 5
      2.3.1 Must Have 6
      2.3.2 Should Have 7
      2.3.3 Could Have 8
      2.3.4 Won’t Have 8

3 Project Methodology 9
   3.1 Process Strategy 9
      3.1.1 Scrum 10
      3.1.2 Roles 10
   3.2 Planning 11
      3.2.1 Initial sprint planning 11
      3.2.2 Revised Planning 12
   3.3 Tools 13
      3.3.1 Visual Studio 2012 13
      3.3.2 PhysX Visual Debugger 14

4 System Implementation 15
   4.1 Design 15
      4.1.1 Global 16
      4.1.2 Physics 16
      4.1.3 Graphics 17
      4.1.4 Touch 19
1.2 Platform ......................................................... 20
  4.2.1 Windows 8 .................................................. 21
  4.2.2 Touch table ............................................... 22
  4.2.3 DirectX 11/HLSL ........................................... 23
4.3 Libraries ..................................................... 23
  4.3.1 OGRE ....................................................... 23
  4.3.2 PhysX ...................................................... 24

5 Quality Assurance ........................................... 25
  5.1 Process Quality ............................................. 25
    5.1.1 Documentation .......................................... 25
    5.1.2 Version Control ......................................... 26
  5.2 Product Quality .......................................... 27
    5.2.1 Functional Quality ..................................... 27
    5.2.2 Stability ................................................ 28
  5.3 Testing ..................................................... 29
    5.3.1 Testing Process ......................................... 29
    5.3.2 Reasoning ............................................... 30
  5.4 Feedback SIG .............................................. 31
    5.4.1 Summary ............................................... 31
    5.4.2 Improvements .......................................... 32

6 Evaluation .................................................... 34
  6.1 General .................................................... 34
  6.2 Planning .................................................... 35
  6.3 Libraries .................................................... 37
    6.3.1 OGRE .................................................... 37
    6.3.2 PhysX .................................................... 37

7 Conclusion ................................................... 39

A SIG Feedback .................................................. 41

B Project Assignment ......................................... 43

C Software Manual ............................................. 46
    C.1 Gestures .................................................... 46
    C.2 Configuration File ....................................... 49

D Software Development Plan ................................ 52
# Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>DLL</td>
<td>Dynamic Linked Libraries</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphics Processing Unit</td>
</tr>
<tr>
<td>GLSL</td>
<td>OpenGL Shading Language</td>
</tr>
<tr>
<td>HLSL</td>
<td>High-Level Shader Language</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>MoSCoW</td>
<td>Must Have, Should Have, Could Have, Won’t Have</td>
</tr>
<tr>
<td>OGRE</td>
<td>Open Source 3D Graphics Engine</td>
</tr>
<tr>
<td>PVD</td>
<td>PhysX Visual Debugger</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SIG</td>
<td>Software Improvement Group</td>
</tr>
<tr>
<td>SPH</td>
<td>Smoothed Particle Hydrodynamics</td>
</tr>
<tr>
<td>XML</td>
<td>EXTensible Markup Language</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The Inspiration and Innovation Lab Delft, or i-Lab for short, tries to improve everyday life. It does this by providing a place where people can prepare for the new municipality, try new ideas and technologies and easily collaborate with coworkers. This project aims at demonstrating how a large multi-touch screen can support creative processes by providing an example of a collaboration tool for the employees of the Gemeente Delft.

1.1 Problem Description

Innovation is an important aspect in any work environment and it can only be achieved with creativity. Therefore, creativity in a work environment has to be encouraged and several approaches can be made to stimulate the creativity of the personnel. One such approach is a dedicated space where employees can unleash their creativity, either together or alone. The multi-touch table at the i-Lab of the Gemeente Delft is one such method of fostering creativity in a team. But the table itself is not enough, there has to be software that encourages cooperation and stimulates the team to work better as a whole.

A problem where such an application could be useful is in seeing where there are problems with water flows and water drainage. Every city in the world has to deal with these problems, but here in the Netherlands these problems are even more important. The city of Delft lies beneath the sea level, with for example the EEMCS faculty being 1
meter below the sea level. Identifying problems with water flows and water drainage before they actually occur is critical.

1.2 Proposed Solution

By combining the solutions for the problems described in the previous section, this project aims to demonstrate how a large multi-touch screen can support creative processes. By looking at the two problems two requirements can be found. The first requirement is of being able to express ones creativity, ones creative thinking, inside an application. This requires the proposed solution to allow the user to do anything the user might want to do in the context of water simulation. The second requirement is that the user should be able to simulate different scenarios concerning water (possibly rainfall or a dike breach) for the Gemeente Delft. By adhering to these requirements in the actual design of the program an effort is made to solve the given problems.

1.3 Outline

The rest of the document is structured as follows. The first chapter after this introduction will describe the assignment in detail, including the requirements of the program. Then, chapter two gives an overview of the project methodology, including the process strategy, the planning and the tools that were used during the project. Thereafter, the third chapter describes the system implementation including a detailed design of the program, the platforms on which it is running and the libraries that were used during the development of the program. The following chapter will explain the quality assurance methods that were put into practice, starting with the product quality, the process quality, the testing and finally the feedback from SIG. In the second to last chapter the development process will be evaluated and a conclusion brings the report to a close.
Chapter 2

Assignment & Requirements

In this chapter, an outline of the assignment and the program will be given. It will provide a good understanding of the motivation behind this project and the program as result of the project. To this end, the first section will give a detailed overview of the assignment and the motivation behind it. The first section will also describe the proposed system on the basis of those motivations. The following section will give a global overview of the requirements that the program as a result of the assignment must meet. A more comprehensive requirement list for the program is presented in the last section.

2.1 Assignment Description and Proposed System

The motivation behind this assignment is twofold. Firstly, the Gemeente Delft wants to encourage teams to bundle all the creativity of the individual members into one by working together on an application that is specifically designed to run on a multi-touch table. By using the innovative and unique features of such a table to the full extent, the simulation can provide experiences that are not possible on normal personal computers or even on tablets. Secondly, the Gemeente Delft wants a program that visualises the effects of a (rain) flood on the city of Delft and how a change in terrain elevation will effect the water flows through the city.

To achieve the latter goal, a simulation must be made that that consists of a world modeled after a heightmap of the city of Delft. Every elevation will be consistent with
the elevation at that point in the actual city to provide an accurate world in which the water simulation can be run. Changing the elevation of the terrain or changing the intensity of the rainfall will allow a user to see the effects this has on the water flows in the entire city. This information can be used to see where there are problems concerning the water drainage and water flows in the city of Delft and what can be done to alleviate those problems.

To achieve the first goal within such a simulation, the program makes sure that every person that is standing besides the table has a perfect view of the modeled city of Delft and that everyone can interact in the same manner, no matter where they might be standing. Everyone therefore has the same information at all times which ensures that every member of the team can contribute equally because there is no inherent hierarchical flow of information.

The simple touch interface breaks down barriers that are so often found in desktop applications where users have no idea what they can and cannot do and are therefore afraid to interact with the system. This stifles their creativity because they can only think about what kind of effects their idea will have instead of getting immediate feedback due to interacting with the program. Therefore, because every member of the team can interact with the application concurrently, a member of the team does not have to explain his idea to the rest of the team, he or she simply shows the idea to the rest of the team. The simulation immediately adapts to the new situation and the rest of the team can observe the impact it has on the city and provide feedback to improve upon the idea.

2.2 Global Requirements

Setting requirements for the program on a global level will give a quick and clear overview of what the program must adhere to. The motivation behind these requirements will give an idea as to why these requirements are crucial to the success of the program. It will also put the more specific requirements presented in the next section into context. The global requirements for this program are as follows:
• The program must display an accurate representation in 3D of the heightmap of the city of Delft.
• The simulation of water in the program is accurate
• The user can interact with the program from any position next to the table.
• The program responds accurately according to the provided input by the user.

If one of these requirements would not be met, the program would not be able to satisfy the assignment as laid out in the previous section. If there is no accurate representation of the city of Delft or if the simulation of water is not realistic, no useful information can be gained from the program. The places in the program where there are problems with the water flows would not be consistent with the places where such problems would occur in the city of Delft.

If the user could not interact with the program from any position next to the table, the program would not use the most important benefit of a touch table. It would not be an application that was specifically designed for a touch table. If the last requirement would not be met, the program would be unusable. This requirement is crucial because many users could interact with the program at the same time. The program needs to accurately respond to all those inputs or else it would not be of much use for a touch table.

2.3 List of Requirements

This section will present the complete list of requirements the program must meet. The list is formatted according to the MoSCoW method, which gives a priority to every requirement to make it clear how important the requirement is. Due to the large list of Must Have requirements, these are also separated according to what type of requirements they are.
2.3.1 Must Have

- **Architectural**
  - Use the Ogre game engine.
  - Use the programming language C++.
  - Use the physics library PhysX

- **Functional**
  - The program simulates the flow of water in a realistic way.
  - The user can deform the terrain of a 3D environment with his or her fingers.
  - First the user selects a piece of land by drawing over the land he or she wishes to select.
  - When the user then makes a two finger double tap gesture inside selected terrain, than that part of the terrain will be elevated.
  - When the user then makes press and double tap gesture inside selected terrain, than that part of the terrain will be lowered.
  - The user can make it rain in a specified area.
  - The user can make it rain more or less in real time.
  - When the user makes a clockwise rotating gesture, it will start to rain more.
  - When the user makes a counter clockwise gesture, it will start to rain less.
  - The user can move the camera by panning two fingers across the screen.
  - The user can zoom in or out by performing a pinching or expanding gesture respectively.
  - The buildings will influence the flow of the water in a realistic way.
  - If there is text on screen, it is displayed multiple times to make sure that it is readable from two sides.
  - The simulator can work with height maps provided by the Gemeente Delft.
  - The desired accuracy of the simulation can be changed before loading a heightmap.

- **Non-functional**
  - The program will be compiled into a 64 bit executable.
• Behavioral
  
  – If a fatal error occurs, the program will exit to the desktop.
  – Otherwise, the program will attempt to continue running without user having to reboot.

• Performance
  
  – The program registers touch input within 1 second when acquiring input from a Samsung TM55LBC touch overlay.
  – The program runs at 30 frames per second at the very least when running on a HP Z400 workstation.
  – The program boots within 10 seconds.
  – The program displays a loading screen or splash screen within 2 seconds.

2.3.2 Should Have

• The program runs at 60 frames per second at the very least when running on a HP Z400 workstation.
• The user can place a storm drain in the world which will act as a sink, removing water from the world.
• When a user draws a shape, that shape will be turned into an object in the world which interacts with the water.
• The user can select which parts of the heightmap are used for rendering.
• The user can select a piece of ground and will the program will then display information about the selected piece of ground. For instance:
  – The amount of water at that location.
  – Size of the area.
2.3.3 Could Have

- The user can draw a line in the world which will create a dike at that location.
- Drawing a line on an existing dike will elevate the dike.
- Realistic environment sound.
- There are other sources of water in the world. For instance:
  - A dike breach.
  - Small scale, local floods due to a water pipe breach

2.3.4 Won’t Have

- The user cannot influence the flow of the water in any other way than manipulating the height of the terrain.
- The program will not account for the saturation of the ground.
Chapter 3

Project Methodology

This chapter gives an overview of the methodology used during this project. Depending on the assignment and on the group of people that are working on the assignment, different approaches can be taken on how to complete the project. In the first section, an outline will be given of the process strategy that the team chose for this project. The reasoning behind this decision will be given in this section as well. The planning that was used for this project will be given in next section. This planning is shaped by the process strategy that was chosen. The chapter ends with a section about the tools that were used during the development of the program. The tools will be described in detail and the reasons behind their use will be given.

3.1 Process Strategy

The work on this project was done by a team of three persons, primarily at the faculty of Electrical Engineering, Mathematics and Computer Science of the TU Delft. When necessary, testing was done at the i-Lab of the Gemeente Delft. An agile programming method is well suited for a small team and a short time to build a product. For this reason, the team chose such an agile programming method. The specific methodology that was used by the team in this project was the scrum methodology.
3.1.1 Scrum

The core concept of the scrum methodology is the sprint. A sprint is a set period of time, two weeks in the case of this project, in which the team commits to developing one or more features of the program. A sprint is started by a planning meeting, where the features for this sprint are selected and the previous sprint will be reviewed. After every sprint, a working prototype of the product has to be available. The global planning of the sprints was closely tied to the deliverables of the bachelor project and is detailed in section 3.2.

Because sprints are fairly short, scrum provides the team the ability to easily adapt to the changes in the requirements of the product owners, which in this case is the Gemeente Delft. In a more traditional approach, the complete design of the system would be done in advance, making it harder for the team to incorporate changes requested by the product owner.

3.1.2 Roles

To make sure that this project will run smoothly, certain responsibilities must exist. To prevent a wait-and-see attitude, specific persons receive clear responsibilities. They are responsible for the part of the project they are leading and should ensure that the part they are responsible for is done properly. In this project the following roles are distinguished:

- **Lead Programmer - Ernst van der Hoeven**: The lead programmer directs the general structure of the program that is being developed. The lead programmer is also the go to for questions regarding the implementation of features in general.

- **Lead Planner - Kevin Kessels (original) / Ernst van der Hoeven (final)**: The Lead planner makes sure that a detailed plan is made available every sprint to make sure that everyone knows what needs to be done. The Lead planner verifies that the planning is being met and takes action if this is not the case.

- **QA Manager - Radjino Bholanath**: The Quality Assurance manager makes sure that everyone in the team pays enough attention to maintain a high standard of coding. Among other things, this includes a proper coding style and that a team
member will only push working code to the repository. The Quality Assurance manager also makes sure that the code is sufficiently tested.

• **Head of Physics - Kevin de Quillettes:** The head of physics has the responsibility of ensuring that the physics are as realistic as possible. He also directs the choices made in the implementation of the physics.

• **Head of Documentation - Kevin Kessels (original) / Radjino Bholanath (final):** The head of documentation is responsible for all documentation regarding the project. This includes the reports and the code documentation. The head of documentation also makes sure that this documentation is consistent, meets the quality standards and that the reports are finished in time.

### 3.2 Planning

This section will detail the planning of the development process. Due to a misunderstanding about the actual assignment, which differed quite a lot from the initial assignment, the orientation phase did not go as planned and this impacted the entire planning. This will be discussed in more detail in section 6.2. The switch of programming language and game engine also impacted the planning quite heavily and this will be detailed in section 3.2.2.

#### 3.2.1 Initial sprint planning

As mentioned in section 3.1.1 every sprint will take exactly two weeks and there should be a working prototype after every such sprint. The planning was made in the first two weeks of the project and is very closely linked to the deadlines of the deliverables. The first sprint will be used to implement the heightmap based terrain rendering and to try out multiple methods of fluid simulation, such as SPH and height field based methods. In the second sprint the water should be rendered and the chosen water simulation should be further improved upon. The terrain editing functionality should also be implemented in part, meaning that only the core features of the terrain editing would be completed in this sprint. The third sprint will include the touch implementation, completing the terrain editor and finishing the water simulation. The fourth and final sprint will be for
bug-fixing and polishing the final product. This last sprint can also act as a buffer for delays from earlier sprints. After the fourth sprint the product will be complete and ready for delivery to the Gemeente Delft.

### 3.2.2 Revised Planning

After switching from C# to the C++ programming language and from the MonoGame game engine to the OGRE game engine the entire planning had to be redone. Because almost two weeks were spent on programming in C# and none of the code could be reused in the new program, some time was lost and the new planning reflects this. The new planning was as follows:

- **First sprint (May 20th - June 3rd)**
  - Basic rendering of the water
  - Basic rendering of the terrain
  - Real time adjustable rain emitter
  - Link the program with PhysX to simulate the water

- **Second sprint (June 3rd - June 17th)**
  - Terrain editing functionality
  - Touch input

- **Third sprint (June 17th - July 1st)**
  - Advanced rendering of the water
  - Advanced rendering of the terrain
  - Write the final report

- **Fourth sprint (July 1st - July 8th)**
  - Improve code according to SIG feedback
  - Make the final presentation
  - Finish any features that are not yet complete

The first priority was actually seeing the terrain and seeing the water particles. At the same time, the link with PhysX needed to be made and the rain emitter had to
be implemented so that particles could be spawned. If no particles could be spawned, the link to the simulation could not be properly tested. All these features had to be implemented before work on the other features could start. Therefore these features were chosen for the first sprint. After those features, the most important feature were the terrain editing and the touch input. Work on those features could be done in parallel, with the terrain editing being done via keyboard commands until the touch input was finished. After work on those feature was done, attention could be shifted to improving the graphics of the program. Both the terrain and the water could look much better than they did after the first sprint. The final report also had to be written in this sprint as the deadline for that deliverable was July 1st. The very last sprint would only last a week due to the date of the final presentation. In this week, the final presentation would be made and any unfinished features would be finished. The code also had to be improved according to the feedback from SIG and this would also be done in this final week.

3.3 Tools

Tools can make the life of a programmer significantly easier, but choosing the right tools for the right job is extremely important. Using the wrong tool can hinder the development process instead of helping it. During this project the development team made use of several tools and this section will give a short overview of all these tools and why the team deemed them helpful.

3.3.1 Visual Studio 2012

Visual Studio is a powerful IDE for several languages including C++, the language the program is coded in. The IntelliSense system built into Visual Studio 2012 saves time by giving auto completion popups, notifying the user of syntax errors and much more. It also provides powerful debugging tools which cuts down the time that is needed to find and fix bugs. Breakpoints can be used to stop code execution so the variables of the program can be analysed, making it very easy to find bugs. These features of Visual Studio made the development much more efficient, by reducing the time the
programmers needed to spend on tasks such as fixing syntax errors, looking up method arguments and debugging.

### 3.3.2 PhysX Visual Debugger

When using a library, debugging something that goes wrong inside such a library is usually quite complicated and unwieldy. For open source projects, the team can compile the source code of the library with the project instead of just the DLL files at a performance loss. This would allow debugging to occur inside the library as well, saving a lot of time. For closed source projects however, such a thing is not possible. Fortunately, PhysX has its own native debugger called the PhysX Visual Debugger or PVD for short. The PVD captures a certain amount of frames (specified by the user) and allows for debugging of every PhysX object in those frames. It displays the shapes, particles and other internal scene objects in a convenient visual manner to allow for easy interaction with the objects in the scene. A screenshot of this visualization is displayed in Figure 3.1. The PVD allows a programmer to see the internal variables and flags of every PhysX object and the programmer can then step through every captured frame to see how they change over time.

![Figure 3.1: The PhysX Visual Debugger. The screenshot shows the PhysX heightfield and some water particles, a few are on the ground but most are in the 'rain cloud'.](image)

Because of the abstract nature of the PhysX interface and the complicated nature of the internal physics simulation, the PVD was an invaluable tool for the team. It saved the team a lot of time when something went wrong with the water simulation or the PhysX objects that interacted with the water.
Chapter 4

System Implementation

This chapter will detail the underlying design and the actual implementation of the program. The first section will give a global overview of the design and will then elaborate upon the design of the graphics, physics and touch components. At the same time, details about the implementation will be given as well. The next section will detail the platforms the system is running on and how those platforms influenced the design decisions. The final section will describe how the libraries shaped the overall design of the system.

4.1 Design

The design of the program is one of the most important aspects of any program and this program is certainly no exception. A good design will make it significantly easier to actually implement the program. A good design will make sure that every team member knows exactly how the program will have to be implemented and will eliminate many misunderstandings. This section will detail all the particular components individually to give a complete image of every part of the program. The implementation details will be given as well to give a idea as to how the design was actually realised. First, however, the global design of the program that combines all the individual components will be outlined.


4.1.1 Global

On a global level, the program is developed to display a model of the city of Delft and to simulate how water flows throughout the city. Interaction with the program will allow a user to elevate or lower the terrain at specific points. The simulation will respond in real time and the flow of the water will change according to the change in elevations. The user can specify how accurate he or she wants the simulation to be by specifying the amount of particles the simulation uses. This is further elaborated upon in section 4.1.2. The user can also slightly alter the appearance of the terrain by specifying the relative height of objects in the world and the overall size of the world in terms of length and width of the overall city.

4.1.2 Physics

For the simulation of the water the physics engine PhysX by Nvidia was used. Reasons for the use of this specific physics engine are given in section 4.3.2. PhysX internally works according to SPH, which simulates the fluid in the scene as a collection of particles. A list of particles is created in the program and passed on to PhysX. When the user spawns some water, particles are added to that list. Every particle has a position and a density and when PhysX accesses the list it uses its internal simulation to update the position and densities of every particle. To ensure collision with the model of the city, this model has to be passed to PhysX as well. To do this, a PhysX heightfield object is created which corresponds to the model of the city. Every change that the user makes to the terrain is reflected in the PhysX heightfield object as well. During the simulation, PhysX then handles the collision between the objects and the friction between the surface of the heightfield and the water. This is reflected in the position updates for the particles that collide with or that are on the surface of the PhysX heightfield.

The amount of particles that can exist at any one time will determine how accurate the simulation actually is. If the limit on the amount of particles is low, two things can happen. Either the user cannot spawn a lot of water, leaving only a small amount of water to be simulated in the world. In this situation, the water that does exist in the world will still behave realistically. The other option is that the user can spawn a lot of water, but the water itself consists of fewer particle per square meter. This will make
the simulation much less realistic. This is because a realistic SPH simulation requires a large amount of particles. In SPH the position and density of the neighbouring particles are used in the calculation of the new position and density of every particle. If there are more particles in the neighbourhood, the new position of the particle will be more accurate. More particles will therefore allow the particles to be very small and this will ensure a large amount of particles in the neighbourhood for any particle.

4.1.3 Graphics

The graphical representation of the city of Delft and the water has been developed using the OGRE game engine, which is elaborated upon in [1.3.1]. The model of the city itself is built using the provided heightmap. The user can specify a row, column and height scaling parameter to adjust the length and width of the world and the absolute heights of individual points. Of course, since these are just scaling parameters, any structure will still have a correct size relative to other structures.

The world is viewed from a top down perspective. This perspective ensures that every user has a good view of the city of Delft. A more common perspective in a 3D world such as an isometric view or a free camera would only work for one specific position next to the table. This would defeat the purpose of a large touch screen table, as the largest advantage of such a setup is that a user can interact with the table from any position next to the table. If a user has a weird view of the world, they cannot properly see what is happening and interaction with the program would be very awkward. Lighting and shading will ensure that even in this perspective, objects are still clearly visually separated from each other. Additionally, the colour of every pixel depends on the height of the terrain belonging to that pixel. These two effects combined make sure that the city of Delft is instantly recognisable from this top-down perspective and that the user can see at a glance what the elevation of the terrain is at any point.

In terms of the water, every particle is rendered as a small sphere. Even though the spheres are quite small, they are still quite a bit larger than a particle in PhysX. Therefore, the sphere belonging to a particle has a position which is slightly higher than the actual particle position. This is because otherwise it would seem that the water droplets would sink into the surface a bit, while they are actually on top of the surface in the
PhysX simulation. A screenshot of the simulation, showing some water on the terrain can be found in Figure 4.1.

**Figure 4.1:** A screenshot of the simulator showing the terrain and some water that is flowing through it. The large building on the right side of the screenshot is the EEMCS faculty.

The terrain and water rendering are primarily done via programmable shaders. The language that these shaders are written in is HLSL and the choice of this shader language is elaborated upon in section 4.2.3. Using programmable shaders for the terrain was crucial in ensuring a smooth running simulation. The other option was to render the terrain in a more traditional way, using polygon meshes. A mesh is a collection of vertices and edges which together form faces. These faces define what the object looks like. The object can have any shape or form by using different vertices or edges. Using a mesh for the rendering of the terrain would be quite logical. However, it would also
be too slow to be actually useful in the simulation. It would take several more seconds to load and show the terrain using a mesh than it would using shaders. This was not a real problem however, as the boot time of the program was still quite reasonable. The real problems came into play when the user was trying to edit the terrain. Any change in elevation, however small, had as effect that the entire mesh had to be recalculated based on the new height values. This took several seconds and the user would have to wait to see the effects of his input. This was very frustrating, because the user wants to see his changes in real time. Another problem occurred when two users were editing the terrain at the same time. The program would then have to build two entirely new meshes for every change, which would take an even longer time. On the other hand, shaders calculate the height of every pixel again and again for every frame. If the user changes height values for some pixels, rendering still takes exactly the same amount of time as it does every frame. Moreover, if two users are editing the terrain at the same time, those height value changes are immediately passed on and the shader processes the changes in the same frame.

### 4.1.4 Touch

For reasons explained in subsection 4.2.1 WM_POINTER was used as touch input to the simulator. When using this touch input it is required to implement manual gesture recognition. This includes for example recognising rotating gestures or swiping gestures. Those recognised gestures have to be translated to their respective actions in the program. Panning with two fingers is used to move the camera, however the recognised pan gesture does nothing by itself. To come to an actual action, the gesture needs to be tracked. The camera movement is then executed based on the movement of the two fingers.

Thus, there is a clear separation of gesture recognition and gesture handling. This separation is also visible within the program code. First, a custom gesture recogniser handles the raw pointer data after which it returns its current view on the pointer. There are a few requirements for any gesture recogniser and there are some specific requirement for the use of this particular touch table. In general, a gesture recogniser should behave intuitively and recognise the supported gestures. All of this should happen in a timely matter. More specific for this touch table is that the distances that are
used to recognise gestures might differ quite a lot from, for example, a tablet device. The target touch table has a relatively low amount of pixels for the size of the table. This means that a pointer position change (in pixels) would mean a greater distance change than the same amount of pixel movement on a tablet. For the touch table, this means that movement recognition distances will be relatively small. A correct setting for such distances is important to ensure a responsive feel of the program. If the program recognises movement too slowly, actions based on moving a finger will have a lengthy delay.

The gesture recogniser does nothing besides responding with its judgement on what a pointer might be doing (in other words, what kind of gesture the user is making). Based on this judgement, the gesture handler implements any action call. A two finger pan is used to move the camera and a one finger pan is used to draw areas that can be lowered or heightened. In this part of the program there is a clear link to the complete program. Input can control the camera movement, but it can also control terrain editing or the fluid simulation. The input that the gesture handler receives consists of nothing more than some coordinates. The gesture handler has to implement its own tracking of this pointer to decide how a certain gesture is used, or where it is used. Not only does it need to track this data, but it also needs to decide whether or not to execute a certain action based on all currently active gestures. Take for example a zooming gesture, this should obviously result in a zoom action in the program. However if a area is being drawn so that it can be edited, a zoom action might be very inconvenient. A drawing gesture first needs to be completed before zoom or camera movement in general is allowed.

4.2 Platform

Every system targets some specific platforms. The most important choices of platforms are the operating system(s) the program is supposed to run on and, in this day and age, on what type of device the program would run on. This last criteria is very important because it determines what type of input the application can expect. A normal desktop computer would send mouse and keyboard input, a tablet would send touch input or something like a Microsoft Kinect would send image data. In this project, the program would only run on the computer connected to the touch table at the i-Lab of the Gemeente Delft. This meant that only one specific operating system had to be
supported and only one type of input, touch input, could be expected by the program. In this section, the effects these platform choices had on the design and implementation of the program will be described.

4.2.1 Windows 8

In terms of the operating system, the touch table at the i-Lab of the Gemeente Delft is running on 64-bit Windows 8. This had some advantages and some disadvantages for the development of this project. These advantages and disadvantages were mainly related to the touch input. This was to be expected, since touch input is something the Windows 8 operating system is primarily designed for. This in contrast with the previous Windows operating systems such as Windows 7.

The primary advantage of Windows 8 is the addition of the WM_POINTER messages to handle touch input. In Windows 7 the only available touch messages were the WM_GESTURE and the WM_TOUCH messages. The new addition, WM_POINTER, is far superior for the purposes of this program. The team first tried implementing the old WM_GESTURE scheme to provide backwards compatibility with Windows 7 computers. This could come in handy because testing could then also be done on the Microsoft Surface table (an old version of Microsoft PixelSense, more on this in section 4.2.2) at the InsyghtLab of the university. However, testing revealed that WM_GESTURE was far too slow to be of any use. Drawing a line quickly would mean that only part of the line would be registered and a quick swipe would often not be recognised at all.

The other alternative for backwards compatibility with Windows 7 computers was the WM_TOUCH messages. However, these messages are far too low level to be of much use. All the gesture and manipulation recognisers have to be implemented by the team themselves from lower level touch events. This would be far too much work when a superior alternative was available. The WM_POINTER messages are quick and efficient and track a gesture completely. This in contrast with WM_TOUCH, where no information is given as to what touch point belongs to which gesture. Another advantage when compared to the WM_GESTURE messages is that WM_POINTER can handle any amount of concurrent gestures compared to the two concurrent gestures of WM_GESTURE.

Another advantage is the built in simulator on all Windows 8 computers. In this simulator, one can run an application and provide the application with all sorts of touch input.
This means that anyone with a Windows 8 computer, but without a touch screen, can still test if the touch implementation is working correctly. This was extremely helpful, because this meant that the team was not forced to go to the i-Lab or the InsyghtLab at the university to test the touch input for the application.

A disadvantage tied to this last advantage is this, the touch simulator is not available on Windows 7 machines and considering that the majority of the team worked on such machines, it left the task of implementing the touch input up to one person which has a Windows 8 laptop. When problems arose during the implementation of the touch input, other members of the team had to work on the sole Windows 8 computer as well. There was no way for another member to look at the problems in his own time and on his own computer.

4.2.2 Touch table

Even though the touch table at the i-Lab was running Windows, it was not a Microsoft PixelSense (formerly known as Microsoft Surface) table. Such PixelSense tables are specifically designed and manufactured by Microsoft as touch screen tables. The device is all in one, meaning the case houses the screen as well as the systems internal components. The touch table at the i-Lab however, is a horizontally positioned TV screen with a Samsung TM55LBC touch overlay on top. The screen and overlay are connected to an external workstation. This setup differs from a PixelSense table in some important ways. The most important distinction is that the touch overlay can only recognise 6 touch points at the same time, significantly less than the 50 or more touch points the PixelSense table can recognise. This means that the application can not require any one user to make gestures with more than three fingers. This would not allow a second user to make the same gesture, but at a different spot. Limiting the number of touch points in a gesture to a maximum of three ensures that at least two persons can interact with the application at the same time. This is crucial for a touch table, since the main advantage of such a setup is interacting with the table alongside other people.

Another difference is that the touch overlay works slightly differently than a touch screen. Touch screens come in two different forms, capacitive and resistive. Capacitive touchscreen use the electrical properties of the human body to detect where a user touches the screen. Resistive touch screens rely on pressure to determine where a touch
point is. The touch overlay on the other hand uses infrared technology to detect touch points. But this means that not only fingers are recognised as touch point, but any other object that hovers a little above the screen is recognised as well. This creates problems when for example a sleeve of someones shirt is hovering just above the screen but is still picked up as a touch point.

4.2.3 DirectX 11/HLSL

A large part of the graphics work of the simulator had to be done via programmable shaders. These shaders are used for the rendering of the terrain and the rendering of the water. Many of the example shaders of Nvidia and the OGRE forums were built using HLSL and for this reason the decision was made to program the shaders in the HLSL language as well. This decision meant that the program had to use the DirectX API from Microsoft for the graphics rendering. And because the DirectX API only work on Windows machines, this meant that the program itself would also only run on Windows machines. An other shading language such as GLSL would allow the OpenGL API to be used and the Cg language would allow both aforementioned APIs to be used, allowing the simulator to be run on other operating systems such as Linux. However, because there is only one target computer for this project (the computer connected to the touch table at the Gemeente Delft) and this computer runs on Windows 8, there was no problem with any program incompatibility.

4.3 Libraries

To help facilitate some of the features of the program, several libraries were used. These provided an implementation of some necessary component of the system and provided an interface to invoke those implementations. This section will detail which libraries were used during the project and why they were chosen.

4.3.1 OGRE

Because the simulation had to model the city of Delft, a game engine was used to simplify much of the work in terms of the graphics and visualisation. Working directly
in OpenGL or DirectX would be far too time-consuming considering the short duration of the project. A game engine provides a rendering engine which makes it significantly easier to make and manipulate complex graphical objects because the rendering engines manages the graphical primitives composing the complex object itself. OGRE also makes it easier to use lights, shadows and ray casting because the user does not have to worry about all the view matrices and most of the transformations necessary to set up proper lighting.

4.3.2 PhysX

An other important aspect of the program is the water simulation. It is critical that the simulation of the water is realistic, efficient and scalable. But implementing such a fluid simulation from scratch via a method such as SPH or a height field based approach would be very time-consuming due to the extensive knowledge of physics necessary to understand and implement such methods. A physics engine manages the simulation internally and provides methods to access the information pertaining to the simulation. This makes it easier to simulate the water because the external physics engine does all the heavy lifting. The physics engine chosen in this project is PhysX by Nvidia. This engine was chosen because it is fast and realistic. Furthermore, it has extensive documentation and a large active user base. This certainly helps when problems arise with the water simulation. Another helpful aspect was the PhysX Visual Debugger, which made debugging the simulation much easier. The PhysX Visual Debugger was elaborated upon in section 3.3.2.
Chapter 5

Quality Assurance

In this chapter the quality assurance regarding the process and the product will be outlined. Starting with an overview of how the process standards are maintained, the quality assurance for every step in the development, from planning to testing and feedback will be explained. After this section, the next section will be dedicated to the product quality. This section is then followed by a section about testing and the chapter ends with a section explaining how the feedback from SIG was processed.

5.1 Process Quality

This section will focus most of all on how the team ensured that the development went as smoothly as possible. The two main factors that played an important role during the development are documentation and version control. Documentation helps to quickly understand certain functionality and version control allows for smooth collaboration between team members.

5.1.1 Documentation

In the section introduction it has been noted that documentation helps to understand certain functionality. Beside the obvious advantage of being able to more easily understand functionality written by someone other than yourself, there is also the advantage of thinking about what you have written. In general, when writing documentation for
code, a clear understanding about what happens where and for what reason is needed. This can force programmers to rethink their code and possibly refactor this code to make it better. For both reasons all functionality in the program has been documented.

For documentation a XML style has been used. This allows Visual Studio 2012 (which is the used code editor and compiler) to give information about functions when they are being typed. When typing a function name, Visual Studio provides the programmer with function information and parameter names, allowing the programmer to continue working without having to look up the function definition. Not only can Visual Studio give information using the XML comments when programming, it is also possible to generate a XML file that can be styled and then viewed in a web browser. This allows for an easy creation of a online API for the source code.

5.1.2 Version Control

As source code management system git has been used. More specifically GitHub\(^1\) which provides free repositories that include an issue tracker and a wiki that have been used to track the issues that arise and to store information in a centralised manner. By using git and GitHub the source code is available from everywhere, and can be edited by multiple collaborators at the same time. Not only is the source code available from everywhere, using git also opens the possibility to work on features in separate branches. By working in separate branches someone can focus on his or her own work without having to worry about changes made by other developers.

At all times there is one master branch that always contains a working version of the system. Only code that can be compiled and actually works is allowed to be pushed to this branch. Features that are incomplete can be pushed but should not introduce bugs or instability. Features, like for example realistic water flow, have been coded in a separate feature branches that have been merged with master when the feature was completed. All those special branches should stay up to date with the master branch. By having to resolve merge conflicts in the separate branches and not in the master branch, possible faults in the master branch are avoided. By using a master branch and feature branches there should always be a working prototype. As new code is committed to the git repository, the code should be read by at least one person beside the original

\(^1\)http://github.com
coder. By reading the code of someone else the chance of missing logic faults in the code is reduced. Such mistakes could stay undetected for a long time and are usually hard to resolve. GitHub also provides developers with the option to make pull requests. In essence, this is a request to merge a feature branch into the master branch. By using this feature, the the way of working mentioned above was realised.

5.2 Product Quality

The main goal of this project was to create a program that would meet the set requirements. Thus the quality of the final product, concerning the functionality and the stability, was of great importance. How the quality of this product has been achieved during this project, is the subject of this section. The functional quality of the program will be discussed first and after that the stability of the program will be reviewed.

5.2.1 Functional Quality

Functional quality depends on the amount of time spent on the project, on how the priorities were set during development and on the actual quality of the implemented features. The amount of available time to spend on the project is a given. This is not so much because of the amount of hours that have to be spent on the project according to the amount of study points, but because there is only a limited amount of time before any deadline. This means the set priorities are of greater importance for the resulting functional quality.

To ensure that the proper priorities were set, the MoSCoW method was used to recognise the importance of the conceived features. For this project, a large amount of time had to be spent on the bare base of the program. Getting the different components of the system to work together, and to program in such a way that the efficiency was sufficient for the scale of the simulation, was the number one priority. Beside that, it was evident that the program should also be useful in one way or another. Therefore the Must Have list contains basic functionality like editing the terrain and making it rain. Most other functionality has been placed in other categories of the requirements list.
The quality of the implemented features mostly has to do with how the features can be used. Assuming that the functionality works as it should, there can still be different ways for how it is allowed to be used. The feature quality has been mostly guarded by acceptance testing and improving the functionality accordingly. Taking for example the possibility to edit the terrain. This can realised by allowing the user to draw simple shapes like boxes and circles that they can than make higher or lower. This could however be difficult to use. A more convenient way for editing the terrain could be to allow the user to draw anything, like using a pencil, and to higher or lower the drawn on terrain.

5.2.2 Stability

Stability, besides being an indicator for the overall quality of the product, is also a quality in itself. As the program is to be used for creative thinking and for testing various solutions to existing and new problems in a professional environment, it should be usable consistently for a prolonged period of time without interruption. When someone works on a solution for some time, it would be very frustrating if this work would be lost because of a crash. Thus not only does stability indicate proper coding, it also is an important goal especially relevant because of the creative context of the program.

As the application relies on libraries quite heavily, the application mostly contains code to tie it all together in a working and efficient manner. The used libraries are assumed to work as they were intended to work, and stability tests for these libraries have not been made. In general, little of the written code was suitable for testing its output and stability. Therefore, for stability there was a focus on checking the code going into the master branch as thoroughly as possible, making sure no logical errors were finding their way into the master branch. Besides making sure that the code was logically correct, there was a very defensive approach to input to functions. By assuming output of functions and input to functions could be wrong or non existing, checks to prevent unexpected behaviour were necessary.

Furthermore, control over parts of the system or external libraries was grouped and confined to very specific parts of the program. For example all changes to the fluid in the current simulation have to be made in the fluid emitter. The fluid system itself only provides functions for the emitter to add and remove fluid but should not be used
anywhere beside the fluid emitter to keep the control of the water confined to a small part of the program.

5.3 Testing

Testing a product is an important step in any development process. Tests can uncover crashes and bugs in the product. During development, they can often help programmers make sure they did not break anything in the program with their changes. The first subsection will explain the details about what was tested and how this was tested. The second subsection discusses why certain ways of testing were used or not used.

5.3.1 Testing Process

Because the application relies heavily on external libraries and therefore mostly contains code that ties this all together, the main way of testing was acceptance testing. Additionally, integration tests have been made. However, because of the graphical nature of the application they were not automated and have been executed in the visual debugger and by storing separate rendered frames.

During the development of the graphical features like the rendering of the water using a programmable pixel shader on the GPU, separate frame output has been extensively used. For example, the input to the GPU consists of square blue blocks as water. In the pixel shader they are then turned into circles. A human can understand graphical output very well, if of course the output makes sense. Therefore, after the changes the result can be very quickly analysed by a developer looking at the output of the shader.

For integrating PhysX with the graphics implemented in OGRE, the PVD proved to be very useful. The PVD shows how the world is seen by the physics engine. Due to the nature of the used terrain rendering techniques, this should closely resemble the graphical representation rendered by OGRE. Both the graphics and the physics engine need to receive the same information about changes in the environment. The graphics need to show the correct environment and the physics engine needs to simulate realistic physics according to the current environment. Because both systems use different ways of representing their data, changing it in both systems at the same time is not as trivial.
as one might hope. However testing these implementations was possible with the PVD and by looking at the rendered screen; both should represent the same change.

For most cases, acceptance testing was the preferred way of testing. Having a list of requirements and working on a feature that has to implement one or more of those requirements makes it relatively easy to see if the program is working correctly. When working on a feature, one can take the requirement and check by using the program if this feature works. This is possible due to the fact that nearly all actions or elements in the program are graphically visible or should give some graphical response. However, things that may seem correct might still be incorrect. Requirements like realistic water behaviour can be confirmed by looking at the water behaviour. However, this might only seem correct when looking at the water flows. It is possible that the water should have actually flowed in another direction. To be certain about the correct working of such complex systems, that only work correctly with a lot of data, is unfeasible.

### 5.3.2 Reasoning

Even though testing is of great importance, to this program and in general, this program does not contain unit tests. As has been mentioned before, this program relies heavily on external libraries. The program does not only use libraries to make the programming in general easier, the libraries provide entire features without which the product would miss the most essential features. OGRE renders frames based on the input and it is difficult to exactly check the correctness of this output. In PhysX it is technically possibly to only simulate one or a few particles and then read the changes after every simulation step. However, the PhysX documentation notes that particles do not behave realistically in this situation and more generally, particles behave unpredictable when there are only a few particles near each other. Thus, one would need to write unit tests for complex physics involving many elements that are being simulated at the same time and therefore also have to be tracked. Writing such tests for both OGRE and PhysX requires a clear and deep understanding of how these libraries behave. This has been considered, but ultimately deemed infeasible by the team.

This however does not explain why no unit tests have been written for the features that were implemented specifically for this project. The gesture recogniser has been written for this project specifically. The same goes for the gesture handler or the input handler
in general. Work on those fully self made features only started after the basis was laid out. The basis consisting of library integration was not useful to test at that point in time. After this prolonged period of time that was needed to implement to get this all working correctly and working together in one program, the list of features that had to be implemented was still long. The decision was therefore made to focus on getting more features finished.

5.4 Feedback SIG

The Software Improvement Group or SIG provides analysis of code and gives recommendations based on this analysis about improving maintainability. They look at the code complexity in general including for example ‘Unit Size’ being the amount of lines of code in functions. Long functions tend to be harder to understand and maintain. In this section the feedback from SIG will first be summarised and the resulting actions to improve the maintainability will be discussed after that. The complete feedback received from SIG can be found in appendix A. The feedback from SIG is only available in Dutch, however.

5.4.1 Summary

The three points of critique on the source code that was analysed by SIG were ‘Unit Interfacing’, ‘Unit Size’ and ‘Component Independence’. The score based on their analysis was a 3 out of 5 meaning an average level of maintainability. Besides the score and the three points of critique they mentioned the missing unit tests. The reasoning behind the missing unit tests can be found in the previous section.

Their first point was the ‘Unit Interfacing’. The score for Unit Interfacing is based on the amount of code with a higher than average amount of parameters. The example that was given is that in many places in the project code the parameters x and y are used. They are not always of the same type, but it nearly never happens that only x or y is needed. They suggested to change from the x and y as separate parameters to some special type. An example where there was already abstracted to a special type to prevent multiple parameters, is in the gesture recognition and handling system. In those parts of the program, the identifier and actual gesture data is combined as a pair.
The second remark was about the ‘Unit Sizing’ which was mentioned in the section introduction. Some functions are more than 100 lines of code. Understanding such a function from start to end can be more difficult than trying to understand many smaller functions with only a very limited task. SIG recommends to split functions in multiple smaller functions if this is possible.

The last remark was about the ‘Component Independence’. For this criteria, SIG looks at how much code is used only internally in a function and how much code is called from outside the different components. If a component uses parts of other components, changing certain functionality in one component might unintentionally affect the components using it. A given example is that the namespaces Windows and Graphics have a circular dependency. Both of these namespaces rely on one another, resulting in that any change in one of them will most likely affect the other namespace.

### 5.4.2 Improvements

Using all the given feedback, all code concerned will be improved upon by applying the given advice. How much refactoring can be done in the remaining time still has to be seen, but at the very least the places that need to be refactored most, will be changed.

For ‘Unit Interfacing’ a clear candidate for improvements is the constant usage of \( x \) and \( y \) as function parameters. The parameters \( x \) and \( y \) are not always used with the same value type, sometimes they are integers, sometimes they are floats and sometimes they are something else entirely. Because of this, using a template for a 2D coordinate type seems the way to solve this problem. The second regularly used function parameters are the terrain dimensions. The terrain width and height, using the four parameters, \( \text{width} \), \( \text{height} \), \( \text{colScale} \) and \( \text{rowScale} \), that combined together form the actual width and height. These values can be combined in some ‘Terrain Data’ structure.

‘Unit Sizing’ is a general problem found in several parts of the source code and therefore splitting functions has to be done in all those parts. The most important places for splitting functions into multiple smaller functions is when there is duplicate code. Although this is not an issue for this particular code, generalisation might still be possible. Functions with clear separate tasks in them can be split so that those tasks have their own function. In the ‘Create’ function from the Window class it is clear that about half
of the function is used for internal variable setting. The other half of the function is used to create the actual Window object. This function should be split in at least two separate functions. It is also possible to generalise some more in this same function. The actual window size and position is calculated and set. However this is also required for the input handler where the movement of the window is handled. This should be moved to some general window moving function. Some functions include switches mostly found in the gesture recognition and gesture handling part of the program. Some cases have become too large and should be moved to an own function. An example of such a function would be the ‘UpdateValues’ function in the GestureRecognizer. The GT ROTATE case has such a large body that this should be moved to a different function altogether. That function should then only be called in the case of a GT ROTATE.

Based on the feedback mentioning ‘Code Independence’, changes will be made mostly to prevent circular dependencies between namespaces. Namespaces are confined groups with functionality related to each other, and two classes in the same namespace might need access to one another. Within a namespace, effort will be made to reduce dependencies but this might not always be logical. For example the FluidSystem class controls the FluidEmitters and therefore updates those with the time steps it receives itself. The FluidEmitter needs to call the FluidSystem because the FluidSystem controls all the fluid in the program. If the FluidEmitter wants to spawn particles, which it obviously wants, then it needs access to this functionality.

Such dependencies are however often not needed between namespaces. Following the example provided by SIG, it can be noted that the Graphics namespace contains a function to resize the window and its internal camera. However the window should control the Graphics and the Graphics should therefore not set values in the Window namespace. This resizing functionality is being moved from Graphics to the Window namespace. The Window namespace should then let the Graphics namespace know whenever it needs to update its own view of the window size.
Chapter 6

Evaluation

Now that the project has almost finished, the development in general can be evaluated. This evaluation is the subject of this chapter. Firstly, a general evaluation will be given. This includes how did the team experienced the development process, what the team thought went well and what would the team do differently next time. The next section will evaluate the planning. It will examine if the team succeeded in following the set planning and, if this was not the case, why the planning was not followed. Because the libraries played such an important role in this project, this chapter will end with an evaluation of both of the libraries that were used.

6.1 General

In general, the whole development process was a positive experience for the team. Unlike many other available bachelor projects, this project had besides its goal of stimulating creativity, a very clear focus on efficiency and scalability. This focus on freedom for the user, to express their creative thinking inside the program, and efficiency in general proved both to be interesting challenges that required acquiring knowledge, working with new tools and libraries, and obviously spending a lot of time and effort to get to a working solution.

In the first stages of the project, the focus was on listing the possible ways of implementing the program to meet the requirements. Although PhysX was already considered at that time, the decision was made to give implementing a custom water physics system
a go. This decision was made to allow programming in C# which does not work very well with PhysX. The main reason however was to prevent having to include a huge library with a lot of functionality that would not be needed and that by implementing the physics manually, a deeper understanding of the systems used would be acquired. Not very long after development started, the scale of the task of implementing physics became clear. Implementing an efficient water physics engine can be considered as a project on its own and the project switched to using PhysX. Section 6.3.2 gives a more extensive view on how the team experienced the library. Generally this switch was considered a good choice by the team.

The switch from a custom water physics engine to PhysX meant the focus could be switched to implementing features instead of a just a basic need for the program in the water simulation. This, however, meant the planning was affected quite a bit. This will be detailed in the next section. After this change of plans everything stayed stable and the features were starting to get implemented. By using the libraries many features during development were easier to realise. How the team experienced working with those libraries is explained in more detail in section 6.3.

Above all, what went well during this project was the effort put into it. A challenging assignment can make some people lose hope on success. However, considering the amount of time spent by every team member, it was clear that the team members were determined and mostly answered by spending more time a week on difficult problems to solve them and make the features work. The difficulty with such an assignment lies not with programming on itself, everyone in the team could do that. The issue lies with the unknown libraries, unknown ways of programming and efficiency in general. These issues were handled throughout the project by careful consideration and extensive studies of the libraries.

6.2 Planning

As explained in section 3.2 and in the previous section, the planning was altered to accommodate the change from C# to C++, from MonoGame to OGRE and from implementing the water simulation from scratch to using the PhysX physics engine by
Nvidia. In this section, the altered planning will be considered and evaluated. However, first some words will be spent on the previous planning.

The major flaw in the first plan was the assumption that the team could, within reasonable time, implement a water physics engine that would be efficient enough for the purposes of a large scale water simulation. Although clearly not impossible to do, thinking that doing this was feasible was a misjudgement from the team. Due to a lack of experience with working on complex projects, the complexity of this assignment was underestimated. However, this is something for the team to remember and will help in the future when making estimates.

The used planning has been based on the Scrum agile development method. This means that every two weeks there is a deadline and that there should continually be a working prototype. Following this method proved to be very useful because features had to be split up so that they could be finished on time. This meant that functionality made it in to the prototype more quickly.

Even though some features were split into smaller parts to allow for a deadline every two weeks, the team faced the the same problem that came up before. Some features, that could not be split up more than they were, were underestimated in terms of development time and could not be finished within the given development period. This problem was mostly faced during the starting period with the new planning. In this period PhysX and OGRE had to be integrated and these libraries had to be connected so that they could together render and simulate the water simulation. At the start, the iteration was expanded by one week after which a normal two week sprint was used again. In the later sprints it was possible to maintain deadlines by sometimes making features smaller.

Having a working prototype and using feature branches all the time was well received by the team. Some features could introduce bugs during development which could hinder development of other features that might be easy or quick to implement. If those easy features would be developed in the same code base as the feature introducing the bugs, then pinpointing the reason of the bugs might become more difficult. By developing the features in a confined area with a working base, the correct working of a feature could be more easily verified. Only after such verification could a feature make its way into the working prototype.
6.3 Libraries

In section 4.3 the motivations behind the use of the OGRE and PhysX libraries were given. In this section, an evaluation of the use of these libraries will be given. Did the team encounter a lot of problems or did the use of the libraries go smoothly? Would the team use these same libraries again in a future project?

6.3.1 OGRE

Regarding the game engine OGRE, which was used for the rendering of all the models, motivation was given in 4.3.1. The primary motivation was that OGRE handles all the graphical primitives, saving users the trouble of working on such a low level. In the actual development of the program, this was certainly very helpful to the team. OGRE provided many helpful classes such as Billboards, used in the water rendering, and its own camera functions. All those features certainly saved the team a lot of time, so in that sense the choice for the OGRE game engine was justified. OGRE also has a large user base and a complete and thorough API. This meant that the team could always look to the API or the forums when they were faced with problems with an OGRE feature. Also helpful was the automated log which all OGRE components write to. This log gives detailed feedback on what happened during rendering and provides more info regarding errors than the error messages shown in Visual Studio 2012. OGRE is also open source and the code can be viewed on their Bitbucket repository. So when the team wanted to know in detail what a OGRE function did, they could look up the piece of code on their Bitbucket repository. This proved helpful in gaining a more comprehensive view of the OGRE engine and all its pieces.

6.3.2 PhysX

The motivation behind the use of the physics engine PhysX, used to simulate the flow of the water, was given in 4.3.2. As said in that section, such a simulation is extremely complex and implementing and testing it would be very time consuming. It would take as much time, if not more, as implementing the rest of the program. Connecting to the PhysX libraries was very easy, largely due to the comprehensive API and the helpful guide that is available with the PhysX SDK. Some problems arose with the peculiarities
with the PhysX shapes used for the terrain and the particle system used for the water. Both of these PhysX objects required parameters for proper initialisation, but no help was available as to how these parameters had to be set. The API and the guide only gave general information, but of course they had nothing to say about the specifics that were necessary for this project. And because the user base of PhysX largely consists of major companies and larger development teams, no real help was available online. Such large teams keep their communication in house and they do not ask for help on, for example, message boards. The PhysX Visual Debugger, detailed in section 3.3.2, was very helpful with those problems and with some trial and error, all problems could be solved in a reasonable amount of time.

The use of both of these libraries was judged positively by the team. If the team were to implement another program that would need a game engine and a physics engine, they would certainly look to both OGRE and PhysX again as primary candidates.
Chapter 7

Conclusion

The goal of this project was to support creative thinking through a program for water simulation for the Gemeente Delft. By allowing multiple users to interact at the same time, this project tries to stimulate creative thinking in a team environment. For example, by changing the terrain and the environment, the users can try out their ideas and see the effects those changes have on the simulation in real time. Aside from trying to stimulate creative thinking, the program also tries to be a useful testing environment for scenarios involving water, like for example heavy rainfall or a flood.

This project tries to solve the stated problems by providing the user with a realistic water simulation program that allows the simulation of heavy rainfall. By simulating the rainfall on a heightmap of the Gemeente Delft, the flow of water in the city can be analysed. The users have a lot of freedom regarding the editing of the environment. They can elevate or lower the terrain however they like, allowing them to see how this may impact the flow of water. By allowing the users to easily make changes to the terrain they can easily and visually test out new ideas that could solve issues with the water drainage at certain areas of the city of Delft.

Currently, the program allows for the user to change the terrain and the way the rain falls. The simulation of the water uses realistic models, making it a good base for additional development based on the work done during this project. A useful improvement to the program would be to allow the user to select a piece of terrain, after which the program would show some information about the selected terrain. For instance, it could show the average height of the terrain in meters and the average water depth. This would make it
very easy to see exactly at what elevation problems occur. Another improvement would be to incorporate the drainage system of the city of Delft in the actual model. This of course depends on if this kind of information is actually available. If the information is available and is integrated into the current system, it would allow for a more realistic simulation. The program could then allow a user to place a new drain in the world to see what kind of effects this has on the water levels. It would also be helpful if the colour of the sphere depended on the distance between the particle and the camera position. A particle closer to the camera would then have a lighter shade of blue than a particle that is farther away from the camera. In this way, a user can more easily see how much water has accumulated on a certain spot when viewing the world from a top-down perspective.

It is clear that the program in its current state is useful for testing certain scenarios, however with additional features like the ones mentioned above the program could be used for testing more detailed scenarios. The team considers the project to be successful and hopes the simulator will prove to be useful in practise.
Appendix A

SIG Feedback

[Analyse]

De code van het systeem scoort 3 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code gemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere score voor Unit Interfacing, Unit Size en Component Independence.

Voor Unit Interfacing wordt er gekeken naar het percentage code in units met een bovengemiddeld aantal parameters. Doorgaans duidt een bovengemiddeld aantal parameters op een gebrek aan abstractie. Daarnaast leidt een groot aantal parameters nogal eens tot verwarring in het aanroepen van de methode en in de meeste gevallen ook tot langere en complexere methoden. Wat in dit systeem opvalt is dat er op verschillende plekken een coördinaat (met een ‘x’ en een ‘y’ waarde) wordt meegegeven aan een methode, maar dat dit coördinaat twee aparte parameters zijn. Om bij toekomstige aanpassingen duidelijker te maken wat er precies meegegeven moet worden aan deze methodes is het aan te raden een specifiek type te introduceren voor dit soort concepten.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. Binnen de langere methodes in dit systeem, zoals bijvoorbeeld de ‘GraphicsController::CreateTerrainBuffers’-methode, zijn aparte stukken functionaliteit te vinden welke gerefactored kunnen worden naar aparte methodes. Commentaarregels zoals bijvoorbeeld ‘// Copy the vertices to the vertex buffer’ en ‘// Copy the indices for each triangle in
the index buffer’ zijn een goede indicatie dat er een autonoom stuk functionaliteit te ontdekken is. Het is aan te raden kritisch te kijken naar de langere methodes binnen dit systeem en deze waar mogelijk op te splitsen.

Voor Component Independence wordt er gekeken naar de hoeveelheid code die alleen intern binnen een component wordt gebruikt, oftewel de hoeveelheid code die niet aangeroepen wordt vanuit andere componenten. Hoe hoger het percentage code welke vanuit andere componenten wordt aangeroepen, des te groter de kans dat aanpassingen in een component propageren naar andere componenten, wat invloed kan hebben op toekomstige productiviteit. In dit geval valt met name op dat er een cyclische afhankelijkheid bestaat tussen de namespaces ‘Windows’ en ‘Graphics’ (via ‘Window’ → ‘InputHandler’ → ‘GraphicsController’ → ‘Window’). Vanwege de complexiteit worden dit soort afhankelijkheden over het algemeen afgeraden, daarom is het aan te bevelen om deze afhankelijkheden nogmaals te bekijken en eventueel te verwijderen.

Over het algemeen scoort de code gemiddeld, hopelijk lukt het om dit niveau te behouden of te verbeteren tijdens de rest van de ontwikkelfase. Als laatste nog de opmerking dat er geen (unit)test-code is gevonden in de code-upload. Het is sterk aan te raden om in ieder geval voor de belangrijkste delen van de functionaliteit automatische tests gedefinieerd te hebben om ervoor te zorgen dat eventuele aanpassingen niet voor ongewenst gedrag zorgen.
Appendix B

Project Assignment

The original assignment of this project is given on the following two pages. However, the actual assignment is quite different from the assignment on the next two pages. For a detailed description of the actual assignment, please see the final report, namely, section 2.1.
Project Proposal
BSc Technische Informatica

offered by

Gemeente Delft
Represented by Herman de Haan
via Rafael Bidarra & Elmar Eisemann
+31 (0)15 27 82528
r.bidarra@tudelft.nl, e.eisemann@tudelft.nl

Project description

Innovative multi-touch interaction

The Inspiration and Innovation Lab Delft aims at combining technology and creativity in order to improve everyday life, by coming up with novel ideas and solutions for various challenges in our society. This project aims at demonstrating how a large multi-touch screen can support creative processes. In other words, the main focus lies on showing innovative uses of such interactive devices. We devised several potential projects to exploit the technology in an interesting way, but the application can still be refined by the students and even new applications could be envisioned.

Some of the examples include:

1) A social game, where graffiti is added to a virtual rendition of Delft to share images and messages;
2) A virtual clean-up game, where the players collaborate to clean and improve the city that deteriorates over time;
3) A game, where dikes are built to win land and safely extend a virtual city;
4) A virtual painting interface to interactively transform a standard rendering into a painterly artwork by associating gestures to paint strokes.

The final project will be exhibited in the Inspiration and Innovation Lab Delft and accessible to researchers and the general public.

**Project team**

**Supervision**
Elmar Eisemann, e.eisemann@tudelft.nl, 015 27 82528
Rafael Bidarra, r.bidarra@tudelft.nl, 015 27 84564
Herman Tjesse de Haan, Gemeente Delft, hdhaan@delft.nl, 015 26 02248
Appendix C

Software Manual

This appendix serves as a short manual for the program. It describes all the gestures that are recognised by the program in detail, including what action each gesture performs. It will also detail the use of the configuration file.

C.1 Gestures

This program is intended to be used on a touch table. It is therefore completely controllable with gestures. The complete set of gestures provides all the functionality of the program. Some examples of gesture functionality would be panning for terrain editing or pinching to zoom. A complete list of the supported gestures is given below.
<table>
<thead>
<tr>
<th><strong>One Finger Pan:</strong> With a one finger pan the user can mark an area on the terrain. This area will stay marked until the user uses a one finger pan from the inside of the marked area to the outside. Removing the mark for an area does not undo any changes made to that area.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Two Finger Pan:</strong> The user can move the camera position by panning with two fingers. The point of view of the camera is thus altered by moving the camera, which follows the two panning fingers. The camera cannot be moved when a “one finger pan” is in progress.</td>
</tr>
<tr>
<td><strong>Rotate:</strong> A rotating gesture is executed by keeping one finger stationary and moving the other finger around it. Rotating clockwise will increase the rainfall while rotating counter clockwise will decrease the rainfall. By rotating enough times counter clockwise the rain can be stopped all together. By turning a few times more counter clockwise after the rain stopped you can remove all spawned water from the map.</td>
</tr>
<tr>
<td>Gesture</td>
</tr>
<tr>
<td>------------------------------</td>
</tr>
<tr>
<td><strong>Press and Double Tap:</strong></td>
</tr>
<tr>
<td><strong>Two Finger Double Tap:</strong></td>
</tr>
<tr>
<td><strong>Pinch:</strong></td>
</tr>
</tbody>
</table>
**Appendix C: Manual**

### Stretch
Stretching does the opposite of pinching. Just like with pinching, stretching does nothing when a “one finger pan” gesture is in progress.

#### One Finger Hold
Holding one finger for 3 seconds will spawn water under your finger as long as you keep touching that point.

#### Two Finger Hold
Holding two fingers on the screen for 5 seconds will toggle between windowed and fullscreen mode.

**Table C.2:** Images copyrighted by GestureWorks (http://gestureworks.com)

---

## C.2 Configuration File

With the configuration file, the user can change many of the parameters used in the program. This will allow a user to customise the program to his or her wishes. The
configuration file can be found in the main ProjectGaia folder when navigating to the ‘content’ folder and then the ‘configuration’ folder. The configuration file is a .txt file and is named ‘variablesconfig’. A complete list of values that can be changed in the configuration file and a short explanation for each value follows:

- **Physics**: These values pertain to the physics, including the water simulation.
  - **AmountOfParticles**: This controls the accuracy of the simulation as described in section 4.1.2. Refer to that section for details on why this is important and what lower or higher values mean for the program and the water simulation. *Default: 100000*

- **RainArea**: These values pertain to the area and height where the rain will spawn.
  - **MinX**: The bottom left X coordinate of the area where the rain will spawn. *Default: 0*
  - **MinZ**: The bottom left Z coordinate of the area where the rain will spawn. *Default: 0*
  - **MaxX**: The upper right X coordinate of the area where the rain will spawn. *Default: 100*
  - **MaxZ**: The upper right Z coordinate of the area where the rain will spawn. *Default: 100*
  - **Height**: The height that the rain will spawn from. If the height is too low, the rain might spawn inside the terrain. *Default: 50*

- **HeightMap**: These values pertain to the visualization of the terrain.
  - **MaxHeight**: In essence, this is a scaling parameter. The highest possible value in the provided heightmap, which is fully white, will be as high as the value given. The rest of the height will then be scaled according to this value. *Default: 100*
  - **HeightChange**: When the user elevates or lowers the terrain, the terrain is changed by a set value. By changing this value, the user can make the changes more gradual or more abrupt. *Default: 5*
  - **RowScale**: The scaling parameter for the rows of the heightmap. A higher value will mean a longer terrain. *Default: 1*
– **ColumnScale**: The scaling parameter for the columns of the heightmap. A higher value will mean a wider terrain. *Default: 1*

• **Window**: These values pertain to the window of the application.

  – **Fullscreen**: This value determines whether or not the application will start in full screen. *Default: false*

• **LineScaling**: These values will be used to scale the thickness of the line that appears when selecting terrain and for the amount of space that the camera will move in one step. Both the thickness and the camera step depend on the camera height and will scale between the two parameters described below.

  – **Min**: The lower scaling parameter. *Default: 1*
  
  – **Max**: The upper scaling parameter. *Default: 20*

• **Camera**: These values pertain to the camera of the application.

  – **MinimumBuffer**: When zooming in, the camera will try to stay above the terrain. This value will decide how close the camera can get to the terrain. A negative value will mean that the camera can go inside the terrain. *Default: 5*

  – **Maximum**: This value will decide how far the user can zoom out. *Default: 400*

  – **BaseSpeedKey**: The amount of space the camera will move in one step is dependent on the camera height. This value (multiplied by the Min LineScaling parameter) is the amount the camera will move when fully zoomed in. It will then scale according to the LineScaling parameters. This value only pertains to key presses. *Default: 5*

  – **BaseSpeedGesture**: Works in the same way as the BaseSpeedKey parameter. However, this value only pertains to the gestures. This value is best left a lot smaller than BaseSpeedKey. This is because gestures allow for more accurate control. *Default: 0.10*
Appendix D

Software Development Plan

The software development plan of this project is given on the following pages. However, because of the switch from C# to the C++ programming language and the switch from the MonoGame game engine to the OGRE game engine, the software development plan is not entirely consistent with the project anymore. Refer to the final report for details regarding C++, OGRE and PhysX. As explained in section 3.2, the planning has undergone some changes as well. Refer to that section for further details.
Software Development Plan

Project Gaia, Gemeente Delft: Touch
Technische Universiteit Delft 2012-2013

Contact Gemeente Delft: Herman de Haan

Supervisors: Rafael Bidarra
Elmar Eisemann
Hans-Gerhard Gross
Martha Larson

Written by: Radjino Bholanath
Ernst van der Hoeven
Kevin Kessels
Kevin de Quillettes
Preamble
In this software development plan, the problem which the to be developed software will solve will be described in detail. This will give the client, the Gemeente Delft, the team that will develop the software (hereafter referred to as just the team) and the supervisors a complete and clear picture of what the software intends to do and the process used to build the software. Furthermore, this document will detail what is expected of the team and the other parties with respect to the software development process, but also with respect to the communication between the parties involved. A planning is then given to give insight into how the team expects to complete the project.

The project will be done in about 11 weeks, from the 25th of April until the week of July 12th. The first 2 weeks of the project are called the orientation phase. In this phase, the team will design the software and do research into the software and hardware that is used in the project, either in the development of the software or in the deployment of the software. After this phase, the actual development of the product will start. This phase will last until the end of the project, after which the product is completed and a presentation is given by the team to the client, the supervisors and any other person which desires to hear more about the product.

Summary
We have created this software development plan in order to guide the creation of our product in a timely and efficient manner. The final product will be a simulator with realistic water physics and terrain deformation all controlled by multi touch. An analysis of the assignment gives us a list of conditions and limitations that are taken into account during development. We have assigned roles to each of our team members to facilitate responsibility and management. For the development we will be making use of the scrum methodology. This will give us the advantage of being able to react timely to demands of the client and always have a working prototype. We have divided the eight weeks we have to develop the simulator into four sprints of two weeks. During each sprint a part of the simulator will be created and implemented. During the first sprint the heightmap based terrain rendering and fluid simulation will be completed. The second sprint will involve the creation of a terrain editor and finishing up the water simulation and rendering. The third sprint will include the touch UI implementation and completing the terrain editor. The fourth and final sprint will mainly be for bug-fixing and polishing the final product. If necessary it can also act as a buffer for delays experienced during earlier sprints. After the fourth sprint the product will be complete. Halfway through the project we will also provide an update to our client to allow them to inspect the progress and request changes. Lastly, we also created the product and process quality assessment criteria that will allow us to maintain high product quality.
1. Introduction

The Inspiration and Innovation Lab Delft tries to improve everyday life. It does this by providing a place where people can prepare for the new municipality, try new ideas and technologies and easily collaborate with coworkers. This project aims at demonstrating how a large multi-touch screen can support creative processes by providing an example of a collaboration tool for the employees of the Gemeente Delft.

This document contains the software development plan. The development plan contains the procedures and schedules that will be followed during the course of this project and it will be approved and corrected as follows. The software development plan will approved by E. Eisemann and R. Bidarra and we will adjust the software development plan based on the feedback we get from our supervisors.

The rest of the document is structured as follows. The first section after this introduction describes the project, what the goal of the project is and what is expected as outcome of the project. The next section gives an outline of what work has to done to meet the requirements of the project. After this, in section 4, an outline will be given of what will be expected of the team members, who the stakeholders of this project are and what facilities will be needed to complete the project. Lastly, in section 5, the quality assurance of the project will be discussed.
2. Project Description
This section will give a general overview of the project. First, we will discuss the project goal, our contacts, and give an assignment description. Next, an overview will be given of the results and the product. Then the conditions and limitations of the project will be analyzed. Lastly, we will take a look at the risk factors of the project.

2.1 Project Goal
The Gemeente Delft wishes to demonstrate the capabilities of multi-touch screens as they have acquired one recently for display in the Inspiration and Innovation Lab Delft. The goal of the project is to demonstrate how a large multi-touch screen can support creative processes. This goal will be achieved by creating a simulator which can be used by employees of the Gemeente Delft to collaborate on problems and makes innovative use of the multi-touch screen of the Gemeente Delft. This will allow the user to learn about water simulation in Delft while also gaining appreciation for multi-touch.

2.2 Contacts
Our supervisors are Rafael Bidarra and Elmar Eisemann. Rafael Bidarra is associate professor Game Technology at the Faculty of Electrical Engineering, Mathematics and Computer Science of TU Delft. He leads the faculty’s research line on game technology, and supervises various MSc and PhD projects in this area. Elmar Eisemann is a professor at the Technical University of Delft, heading the Computer Graphics and Visualization Group. His interests include real-time and perceptual rendering, shadow algorithms, global illumination and GPU acceleration techniques. Together they advise and supervise our project.

Our client, the Gemeente Delft, is represented by Herman de Haan. He is a geo-information specialist at the Gemeente Delft. Communication with Herman consists of questions regarding the clients needs, desires and information regarding the multi-touch screen.

2.3 Assignment Description
The assignment is to design and create a simulator that demonstrates the capabilities of a multi-touch screen. This is further elaborated upon by specifying that the implementation must use multi-touch in an interesting way. In order to achieve this the capabilities of the screen have to be explored, and compatible development tools have to be acquired. The multi-touch screen that is being developed for is provided by the Gemeente Delft. As we are providing the software for the Gemeente Delft’s hardware there is also an obligation of result.

2.4 Results and the Product
The result of the project will be a piece of software, in the form of a simulator, which demonstrates the multi-touch functionality of the screen. The simulator will include accurate and interesting water simulations, while allowing the user to influence the simulation in innovative multi touch ways. Through this product the user should develop a greater appreciation for multi touch. The resulting product should be interesting to use, people can come in and get started quickly, supporting creativity and creative solutions by means of an easy way to try out different
solutions. They can simulate water on a certain map and try out some alterations to the map to see how it affects the water flows.

2.5 Conditions and Limitations
The product has to be functional and bug-free. It should be able to run smoothly for an entire day in the Inspiration and Innovation lab Delft, without intervention from the staff. During this time users should be able to find an engaging and fun experience that demonstrates the multi-touch capabilities of the screen. The user should also be able to comfortably operate the product thanks to the multi-touch interface. The multi-touch screen is limited to six simultaneous touch points. Proper care will be taken to not exceed this limitation.

2.6 Risk Factors
The product has to properly show off the capabilities of the multi-touch screen of the Gemeente Delft. This is ensured by designing the project from the ground up to make maximum use of the multi-touch functionality. The product also has to be completed on time. This can be jeopardized by unforeseen problems such as hardware failure or changes in the libraries that will be used. Through proper planning and working full-time we will try and minimize the chances of unforeseen problems happening.

3. Approach
This section will discuss the methodology, the techniques and the activities that will be done during the process of creation the product. We take into account the actions that need to be taken to meet the requirements, how the actions fit in the phases of the projects and what assumptions were made during the process of turning requirements to actions. This will give a clear overview of how the process of product creation is approached and will be executed.

3.1 Methodology
The product that has to be delivered at the end of the project will be designed in the first two weeks. During this period there will be extensive communication between us and our contacts about how the final product needs to look and feel. This will be done to make sure that the client, as described in the contacts section, gets what he signed up for.

After the design period the scrum methodology will be used to implement the features of the product. This means that the functionality will be added in bi-weekly sprints and every two weeks a part of the product will be delivered. One of the reasons that the scrum methodology is used is that it allows for changes to the design if the client wishes to change something about the design. The scrum methodology also makes sure that the client has a working, bi-weekly updated, product even when the project is not finished yet. This methodology will therefore be the best for not only the client, but it also helps us to organize the development in a well organized manner.

3.2 Techniques
During the project we will be using techniques that complement the scrum methodology. After
having done the necessary research and conceptualizing we will discuss our ideas with the client and our supervisors. This should refine the design according to their demands before we actually start to work on it. Once this is done, we can start on the scrum guided implementation of the product and its features.

The first task is to create the backlog, an ordered list of requirements. From this list we can obtain the features that require implementation during the next sprint. Then, for each bi-weekly sprint, a portion of the product will be developed and finished. This means that it will be fully implemented and the added features should be accessible in the always functional prototype. Through preliminary meetings the features to be completed for each sprint are obtained from the already created backlog. These techniques allow us to react to the demands of the client as requirements can be added and removed from the backlog as necessary. Since the goals of each sprint are not decided at the start of the project, but at the start of each sprint itself changes made to the backlog will take effect each week at the beginning of the next sprint. This allows us to react timely to the potentially continuously changing backlog.

In order to maintain an efficient form of documentation of the scrum process we will be making use of Google Docs. This will allow us to keep our scrum documentation transparent and up to date for everyone involved in the project, as it updates almost instantly and is accessible as long as one has an internet connection and our permission to view it. Since the entire project group will be working together in the same space for the entire duration of the project communication within the group can be done vocally. Any important changes to the development process can then be reflected in the Google Docs as to maintain an organized and accurate transcript of the proceedings.

4. Project Design

This section will describe the details of where and how the project will be done. First, the individual responsibilities of every team member will be detailed. Second, the requirements in terms of skill and knowledge will be set for the team members. Afterwards, the stakeholders of the project will be detailed. Subsequently, the techniques and facilities used in the project will be described. Finally, the way information will be shared between stakeholders will be presented.

4.1 Responsibilities

To make sure that this project will run smoothly certain responsibilities must exists. To prevent a wait-and-see attitude specific persons receive clear responsibilities. They are responsible for the part of the project they are leading and should insure that their part is done properly. In this project we distinguish the following roles:

**QA Manager - Radjino Bholanath:**
The Quality Assessment manager makes sure that everyone in the team pays enough attention to maintain a high standard. This includes providing unit tests for functionality, proper coding style and sleeping.
**Lead Planner** - Kevin Kessels:
The Lead planner makes sure that a detailed plan is made available every sprint to make sure that everyone knows what needs to be done.

**Head of Physics** - Kevin de Quillettes:
The head of physics has the very important responsibility of ensuring realistic physics that our client would like to see in the final product. He also directs the choices made in the implementation of the physics.

**Lead Programmer** - Ernst van der Hoeven:
The lead programmer directs the general structure of the program that is being developed and the go to for questions regarding implementation of features in general.

**Head of Documentation** - Kevin Kessels:
The head of documentation is responsible for all documentation regarding the project. This includes reports and also code documentation. He makes sure this documentation is consistent and that reports are finished in time.

**4.2 Personnel Requirements**
Every member of the team is expected to commit at least 42 hours per week to the Bachelor Project. Working 5 days a week at the TU from 9 to 5 will sum to 40 hours, so every member is expected to at least perform some work during the weekend or in the evening at home. If a member is absent during working days he will have to compensate for those lost hours by working more during the weekend or in the evening at home. The team member will notify the team of his absence in advance. Of course, every member is expected to put in real effort during those hours when he is working.

Next to this, it is expected that every member of the team has the necessary programming skills to program the simulator in C# with the MonoGame framework and any other libraries or tools used throughout the project. Furthermore, a member should have knowledge of the scrum methods and knows how to put these methods into practice. If a member is lacking in any department, he will have to gain the required knowledge in his own time to make sure he doesn’t waste time of the entire group by having to catch up on skills.

**4.3 Stakeholders**
The client of this project is de Gemeente Delft. The contact for the Gemeente Delft is Henk de Haan, as detailed in an earlier section. De Gemeente Delft wants an innovative multi touch application for a multi touch table to showcase what this table can do. The supervisors of this project, Rafael Bidarra and Elmar Eisemann, were also described in detail in an earlier section. They will keep an eye on the project and will provide assistance if they consider it necessary. The last stakeholder is the team itself. The team is committed to providing an excellent multi touch experience which will make good use of the possibilities multi touch can provide.
4.4 Techniques and Facilities
The simulator has to run on a HPZ400 computer, which has a 256GB SSD, a Quadro video card and 12GB of DDR3 RAM. Touch input will be provided to the PC by a Samsung TM55LBC touch overlay, which is overlain on a 55 inch screen with a resolution of 1920x1080 pixels. The touch overlay can recognize no more than 6 simultaneous touch points. The operating system running on the PC is Windows 8, which does not need any other drivers to work with the Samsung touch overlay. The application itself will be programmed in the C# programming language with the help of the MonoGame framework.

All programming will be done with Visual Studio 2012, which requires members to program on a Windows computer. As will be detailed later, version control will be handled by GitHub, using command line interfaces or Git clients to use the version control system. Reports will be made with Google Docs.

The work will be performed for the most part at the EEMCS faculty of the Technical University of Delft. On the third floor of EEMCS is the InsyghtLab, where members can meet to work on the project. However, the working areas on the third floor of EEMCS and on the Drebbelweg can also be used to work on the project. Working in the InsyghtLab is only necessary when testing on an actual touch table instead of an emulator is needed.

4.5 Information
The team will start every workday with a small meeting where tasks are delegated and concerns or problems will be brought to the table. Because the team works in the same room and is working together, information will be shared in person. If someone is working from home, communication will be done via email or Skype. Communication between the team and the supervisors will primarily be done via mail. If necessary, appointments can be made to meet in person. All relevant information will then be shared well before the meeting to ensure that everyone is informed. The primary way of sharing this information will be via Google Drive. Documents in the Google Drive folder can be accessed at any time by the team and the supervisors. Information in this folder includes, among others, the requirements, the software development plan and the orientation report. Communication between the team and the Gemeente Delft contact will be done via appointments or, if there is a small question, via email. Relevant information will then be shared in advance via email. As requested by the Gemeente Delft, we will give a full update of our progress halfway through the project and at the very end of the project.
5. Schedule
This section will give an overview of what has to be done for this project, how long those things will take and under what assumptions. First the norms and assumptions, on which the decisions for the schedule are based, will be discussed. After that the required activities to come to a working product will be explained. Ending with a schedule that takes into account the activities that have been discussed and the assumptions how long those activities will take.

5.1 Norms and Assumptions
One of the most important norms within the team is that a deadline is a hard deadline. It is possible that because of unforeseen circumstances the deadline becomes undoable but in any other case the deadline will be kept if it was realistic. This means that it is normal that if some deadline turns out to be more work than expected, that the team has to work more hours to achieve the required result.

Another important norm would be that not any one member of the team will settle for less than a good result. This does not mean that what has been done should be perfect but it should not knowingly be not good enough and should not be a shortcut for something that will result in problems later on. If the team has to do something the result should be of such a quality that it can be used to build upon and it has not have to be redone.

Based on past experiences it is assumed that features of the complexity of water physics take about two working weeks to be programmed and implemented. Features are in this project for example manipulating the terrain and water rendering. These features are not standalone features but they are part of a bigger system, in this case the simulator that is being developed. If a feature is completely separate like maybe an external map editor they take less time to implement. It should be noted that a feature can’t be done in those two weeks if features it relies on are not finished. Creating a map for the simulator is a feature but would only be possible in the indicated time if a certain map engine is already built.

5.2 Activity Plan
To create the simulator the following activities have to be executed:

- The team has to get used to the C# programming language. To achieve this a time can be set, there might after this time be more to learn but the basics should be covered within the time. To get comfortable with C# one day for every team member should be enough, in this time practice programmes can be made and information can be read. This activity affects the whole coding part of the project, because C# is the main programming language.

- Realistic physics have to be integrated in the simulator, because the simulator is based on realistic physics and mostly realistic water physics the actual design of the simulator terrain has to wait for this to be completed. Only integrating it in the code and ignoring the actual implementation should require 6 working days, because it is the main mechanic of the simulator, it has to be done to near perfection, therefore justifying additional available time for this integration.

- The simulator is built for a multi touch table and only has touch input. The targeted
platform, Windows 8, supports touch natively, it should therefore be possible to, without additional work, acquire this touch input. Interpreting the touch positions should be done within three working days whereas gestures are features of their own and require on a very global level also three days to implement. For testing on the touch table this multi touch input is the main dependency. The gestures and touch inputs should have certain consequences, for example it should be possible to shape the landscape by gestures.

- After the preliminary work the water and it’s fluid simulation has to be implemented. This will be composed of creating the water physics and animating and rendering the water.
- Another activity is rendering the terrain according to a height map. This activity can be split into two elements. The first part is reading the height map’s useful data, then the terrain has to be generated according to the values obtained from the heightmap.
- The last activity will be the terrain deformation. Realizing this feature will consist of reading user touch input and editing the height map according to this input. Then the terrain has to be animated to change as commanded by the user. The simulator will have responded to user’s touch input and gracefully changed the terrain to reflect the changes.

5.3 Milestones & Product Plan
The milestones and the corresponding product plan are as follows. The first sprint will be used to implement the heightmap based terrain rendering and try out multiple methods of fluid simulation. The second sprint will be the creation of a terrain editor, rendering the water and improving the chosen water simulation method. The third sprint will include the touch UI implementation, completing the terrain editor and finishing the water simulation. The fourth and final sprint will be for bug-fixing and polishing the final product, it can also act as a buffer for delays from earlier sprints. After the fourth sprint the product will be complete and ready for delivery to the Gemeente Delft.

6. Quality Assurance
In this section an outline will be given about how the quality of the process and product will be assessed. This section will start with the requirements which are used to assess the quality of the end product. After this, the quality assessment of the process is discussed in section 6.2. Lastly a description is given of how the quality of the product will be assessed.

6.1 Product Quality
The quality of the product will be assured by setting certain goals for the product. Using a list of requirements, some of which are mandatory and others that are optional, it can be confirmed if the product satisfies those requirements. All those requirements should be measurable so that they can be tested and so that they are easier to implement because the needed values are provided by the requirements. The list of requirements for this project is provided as an appendix and is structured according to the MoSCoW method. The requirements that are a must should under all circumstances be met for the product to pass its quality test. Requirements in other categories like “Should Have” would add value to the product but are not essential to it. At least some of these requirements should be fulfilled to achieve a product that is more than the
6.2 Process Quality

An important aspect of quality assurance is the evaluation of the quality of the process which is used to build the product. For this project we found that documentation, version control of the source code and the required skills the most important aspect of process quality.

6.2.1 Documentation

The documentation for this project will consist mostly of web pages generated by visual studio. In C# code there is a special xml commenting format that can be compiled to web pages. The idea is that the documentation of the classes, functions and possible other systems can be documented in the code where the actual function or class is specified. Then when the code is changed the documentation can easily be changed to reflect the changes made in the code. An up to date documentation can always be generated from the latest source. Classes can get a global summary of what they are used for or what they have to offer to the programmer. Because this project will be built using MonoGame, a graphics library, not everything has to be built from scratch. It should at all times be clear how and why certain features are used and what uses those features.

6.2.2 Version Control

As source code management system git will be used. More specifically GitHub, it provides free repository’s that include an issue tracker and a wiki that will be used to track the issues that arise and to store information that needs to be sustained over a longer period of time. There will be one master branch that should always be a working version of the system. To this branch only code may be pushed that can be compiled and actually works. Features that are incomplete can be pushed but should not introduce bugs or instability. New features, like for example realistic water flow, should be coded in a separate feature branch that can be merged with master when the feature is complete. All those special branches should stay up to date with the master branch. By having to resolve merge conflicts in the separate branches and not in the master branch possible faults in the master branch are avoided. Tests should obviously run without problems before pushing the changes to the master branch. By using a master branch and feature branches there should always be a working prototype. As new code is committed to the git repository, the code should be read by at least one person beside the original coder. By reading the code of someone else the chance of missing logic faults in the code is reduced, such mistakes could stay undetected for a long time and are usually hard to resolve.

6.2.3 Skills

A few skills are needed to work on this project. Most of the coding for this project will be done in C# and graphics will be realized with MonoGame, it is C# based. Before the first programming iteration starts the team has to get used to all the tools and languages that will be used. Java is in many ways very well comparable to C# and because the whole team is experienced with Java the transition regarding the program language should be short and easy but is very important for a smooth programming phase.
Because the project will be built with MonoGame as the graphics library the team should be familiar with most common features in MonoGame, to render and load graphics for example, before starting. MonoGame is basically the same as the XNA API, documentation on MonoGame can therefore be found in the XNA API. It is therefore required that the team knows their way around the available XNA documentation as to be able to program with it effectively and without having to look for examples every time something has to be programmed.

6.3 Evaluation

The simulator being made should be constantly evaluated to ensure conformance with the design of the simulator. Based on the requirements and the design tests have to be made to control several aspects of the program. New functions and classes should have unit tests to ensure expected behaviour. When code is edited or added those tests should pass without problems and additional tests have to be added for the added code or functionality. Beside testing on the code level there should also be acceptance tests. The requirements should provide a clear list of things that can be observed and tested by a human. This list should be gone through after each scrum iteration to see what requirements are met and what is left to do. However the requirements may be insufficient to come to a satisfying result, therefore it should be strived for to meet with the client or our supervisors every other week to show the progress and get feedback. Using this feedback requirements may be edited, the design could change and accordingly the code has to be changed resulting in a better end result.
Appendix A - Requirements

**Must Have**

**Architectural**
- Use the MonoGame game engine.
  - Use the OpenTK libraries to program for the GPU.
- Use the programming language C#.

**Functional**
- The program simulates the flow of water in a realistic way.
- The user can deform the terrain of a 3D environment with his or her fingers.
  - First the user selects a piece of land by boxing the area he or she wishes to select.
    - When the user then makes a two finger drag gesture out of the selected terrain, than that part of the terrain will be elevated.
    - When the user then makes a two finger drag gesture into the selected terrain, than that part of the terrain will be lowered.
- It is always raining in the simulation and this is the source of the water.
- The user can make it rain more or less in real time.
  - When the user makes a clockwise rotating gesture, it will start to rain more.
  - When the user makes a counter clockwise gesture, it will start to rain less.
- The user can move the camera by dragging one finger across the screen.
- The user can zoom in or out by performing a pinching or expanding gesture respectively.
- The buildings will influence the flow of the water in a realistic way.
- If there is text on screen, it is displayed multiple times to make sure that it is readable from two sides.
- The simulator can work with height maps provided by the Gemeente Delft.
- The desired accuracy of the simulation can be changed before loading a heightmap.

**Non-functional**
- The program will be compiled into a 64 bit executable.

**Behavioral**
- If a fatal error occurs, the program will exit to the desktop.
  - Otherwise, the program will attempt to continue running without user having to reboot.

**Performance**
- The program registers touch input within 1 second when acquiring input from a Samsung TM55LBC touch overlay.
- The program runs at 30 frames per second at the very least when running on a HP Z400 workstation.
- The program boots within 10 seconds.
- The program displays a loading screen or splash screen within 2 seconds.
Should Have

- The program runs at 60 frames per second at the very least when running on a HP Z400 workstation.
- The user can place a storm drain in the world which will act as a sink, removing water from the world.
- When a user draws a shape, that shape will be turned into an object in the world which interacts with the water.
- The user can select which parts of the heightmap are used for rendering.
- The user can select a piece of ground and the program will then display information about the selected piece of ground. For instance:
  - The amount of water at that location.
  - Size of the area.

Could Have

- The user can draw a line in the world which will create a dike at that location.
  - Drawing a line on an existing dike will elevate the dike.
- Realistic environment sound.
- There are other sources of water in the world. For instance:
  - A dike breach.
  - Small scale, local floods due to a water pipe breach
- The program will not account for the saturation of the ground.

Won’t Have

- The user cannot influence the flow of the water in any other way than manipulating the height of the terrain.
Appendix B - Program Description

The intended purpose of the program is to visualize the effects of a rain flood on the city of Delft and how a change in terrain elevation will change how the water flows through the city. To do this, the program consists of a world modeled after a heightmap of the city of Delft. Every elevation is consistent with the elevation at that point in the actual city to provide an accurate world in which the water simulation can be run. The city in the program will suffer from a continuous downpour providing a steady stream of water to the streets of Delft. The user can interact with the program in a variety of ways to change the state of the simulation. Multi touch gestures are used to interact with this world. A rotating gesture can be performed to increase or decrease the amount of rain that is falling from the sky. A user can select an area of the map by drawing a box around it. He or she can then perform a three finger drag gesture to change the elevation of the selected area. Pinching and expanding gestures can be made at any spot in the world to zoom in or out. A one finger drag will move the camera so the user can view any desired location in more detail. Changing the elevation of the terrain or changing the intensity of the rainfall will allow a user to see the effects this has on the water flows in the entire city. This information can be used to see where there are problems concerning the water drainage and water flows in the city of Delft and what can be done to alleviate those problems.