# Real-time Procedural Generation of Environments

Bilal Arslan & Patrick Jørgensen


Kongens Lyngby 2014
IMM-M.Sc.-2014-1

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Matematiktorvet, building 303B,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3351
compute@compute.dtu.dk
www.compute.dtu.dk IMM-M.Sc.-2014-1

# Abstract (English)

The goal of this thesis has been to create a technology allowing users to generate levels for a specific game using paper, pen and the camera of a mobile device. The thesis describes the design and methods for extracting and handling input, as well as procedurally generating an environment.

A prototype, which is able to generate levels from drawing on paper, as well as allowing the user play them, has been created. The users having tested the prototype showed great interest, amazement and satisfaction. We have therefore reached our goal.

# Abstract (Danish)

Målet med dette speciale har været at skabe en teknologi som tillader brugere at generere baner til et specifikt spil ved brug af papir, farver og et kameraet fra en mobil enhed. Specialet beskriver det desgin og de metoder brugt til at udtrække og håndtere input, såvel som procedural generering af omgivelser.

En prototype, som er i stand til at genere baner ud fra tegning på papir, og som tillader brugeren at spille dem, er blevet skabt. De brugere, som har testet prototypen, har vist stor interesse, forbløffelse og tilfredshed. Vi har derfor nået vores mål.

# Problem Statement

In this thesis we will look at the following questions:

- How to generate an environment entirely based on a 2D sketch with curves and how do we introduce varying heights to these roads?
- How do we ensure that the procedurally generated environment (roads, hills, trees, buildings, ...) is done in real-time and is customizable in real-time also?
- How can real-time procedural generations of environments in such a way be used in games?

The first main problem statements gives rise to the sub-questions: How do we read from a sketch? How do we turn them into curves? How do we read heights from a 2D sketch? How do we generate the different objects of the environment?

For the second statement, we have to ensure that our problem can be used in real-time and thereby setting us the requirements to make optimized and efficient algorithms. Optimally, the product should run on a mobile device with limited performance.

The motivation behind this project comes from the third statement, which is making this tool usable for games and is also of wide interest to a lot of other people [JT11]. This also rises the discussion of manual level design versus procedurally generated level design. Can we make a tool that can help anyone generate levels for a specific game, no matter the input?

# Preface

This thesis was prepared at the department of Informatics and Mathematical Modelling at the Technical University of Denmark in fulfilment of the requirements for acquiring an M.Sc. in Informatics.

The thesis deals with procedural generation of environments and how we can turn a drawing from a piece of paper into a real-time virtual environment.

The thesis consists of different methods, analysis and discussion used to make a product, which can be useful in games.

Lyngby, 17-July-2014

Bilal Arslan & Patrick Jørgensen

# Acknowledgements

We would like to thank our advisor, Michael Rose for helping us with ideas, references and advice. Furthermore, we would like to thank him for guiding us through problems and getting us on track regarding our project orientation.

We would like to thank the interviewers and user testers Marta La Mendola, Johan Buhl, Peter Buchhardt, Astrid B.Z. Madsen, Mikkel Martin Pedersen, Martin Vestergaard, Konrad Stanek and high school students for helping us with thoughts and ideas and get us motivated on our product.

# Contents

Chapter 1

# Introduction and Background

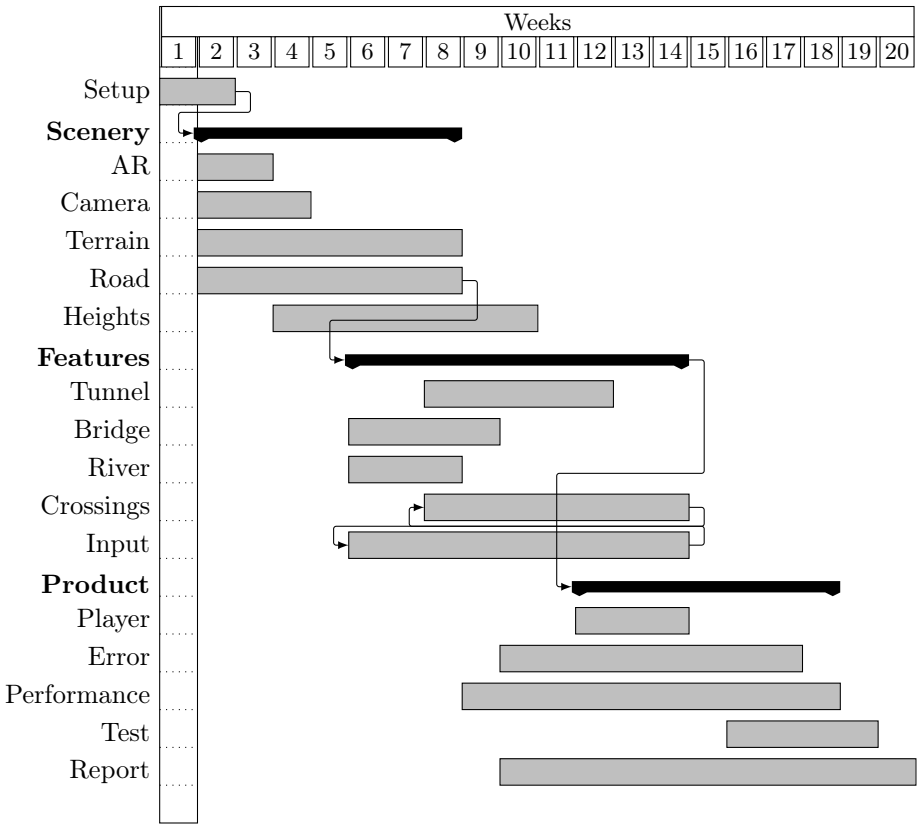
In this chapter, we are going to analyse the making of a tool in Unity with focus on the different aspects of geometry generation, image analysis and run-time problems for generation of a virtual environment for a game. Some of these aspects include mathematical solutions while others focus more on the social aspects such as its usage, purpose and communication.

*What is procedural generation?* By defining a set of rules and measures for an environment, we are able to generate the environment mesh with textures and lighting. It is simply modelling by code, using parameters to define the objects.

The basic idea is to have a camera snapshot of black curves on a white piece of paper, where the curves represent roads. The snapshot is used to extract information required to translate it to a meaningful 3D object for the user. For our case, the black curve would translate into a road or path on a terrain. Furthermore, the idea is to include some features for the environment to make it look realistic and pleasing for the user, such as vegetation and architectural buildings. The product should have a degree of customizability and flexibility.

## 1.1 Project Plan

Here is the main project plan that we followed during our research and development.




The plan is divided into 3 main parts, where the first one, `Scenery`, is the core feature of the procedural generation in our program. The second one, `Features`, encapsulates the additional elements in our scene such as tunnels, bridges, rivers, crossings etc, which 'complete' the scene view. The third one, `Product`, is polishing and finalizing for an end product. Note that this diagram does not take the additional amount of time spent on interviewing level designers into account. (see 1.2)

## 1.2 Project Orientation

In the process of defining our product, we had to go through many steps. Our first idea was to make a level design tool that would allow any level designer to scan sketches and generate fast prototypes to test ideas. This idea came to us after having worked with level designers on DADIU[1].

### 1.2.1 Tool for level designers

Having to make a product for level designers, we had some questions to answer: What is a level design tool? What are its main features? What aspects of level design are slow, and allow us to optimize the process? Do Level designers always start their work on paper? What is a normal creative process of designing levels? How do level designers work? (Appendix A)

In order to find out these things, we decided to make some research online. We looked at the website `World of Level Design`[lev12] on level design in order to understand if level designers sketch levels, what they sketch, and what symbols they use. We found that level designers have their own symbol conventions. They sketch paths and rooms, as well as simple symbols that represent events or pick-ups at these locations.

Knowing little, and having learnt a little more about level design, we decided to interview some professional level designers, as well as some we had already worked with during our DADIU project. This would allow us to get some actual insight as to how real level designers work and think, and help us identify the bootleneck in their working process.

After having interviewed them, we got some great insight in their creative process and how they sketch levels. However we also found out, that what we wanted to make, was not what level designers needed. They are asked to start making levels, once the game is well defined, and a prototype of the game is already up and running. Therefore a simple prototype of the core game mechanics is already made, and therefore a general tool is not of interest.

[1] The national academy of digital interactive entertainment, `www.dadiu.dk`

### 1.2.2 Average people as target

However rethinking the target users of our project, we thought of making a tool which would allow any user, to draw and generate levels. Everyone can draw, not everyone can model 3D objects. So the tool will allow anyone to create levels for a defined game.

This changed the focus of the project. Instead of having to focus on image analysis, and how to identify simple symbols, we would focus on input handling, and usability. This means we can restrict the work of analysing images and focus on getting coherent and good looking output. The aim is to give the user a feeling of "Wow, I did this". Therefore the road that is created, needs to have exactly the same shape as the road that is drawn on paper, and it has to always look good.

It also means, that we have to make many decisions, regarding what should be drawn on paper, and what can be assumed and concluded from what is drawn on paper.

Chapter 2

# State of the art

In this chapter, we will talk about other related projects. We used some of them and took inspiration from others. This lead us to discussions which resulted in our own solutions. The chapter is divided into four parts. The first part is about the level design aspect, where we talk about tools and engines. The second part explains procedural generation in general and its history. The third part discusses Augmented Reality. The fourth and the last part discusses related and similar work.

## 2.1 Level design

Levels in games are generally implemented manually (i.e. building blocks, 3D modelling etc.) by either a Game Designer or a Level Designer. They are using tools such as UDK[1], Unity, Photoshop, Sketch-up[2], 3DSMax[3], and some of them are more into physical toys like Lego, Stick-men or board game pieces. But all of the level designers have 1 thing at common; they are all using paper and pen as their main tool (Appendix A).

[1] Unreal Development Kit, https://www.unrealengine.com/products/udk
[2] SketchUp, http://www.sketchup.com/
[3] 3D Studio Max, http://www.autodesk.com/products/3ds-max/overview

`World of Level Design`[4] is a website, recommended by one of the level designers. It is used to teach about level design and game environment creation. Their focus lies in getting the most out of using the mentioned tools. Their approach lies in creating objects from scratch, but in procedural generation, some rule-sets are defining your creation of objects.

## 2.2 Procedural Generation

`Elite` was the first game to have procedural generation of a world [How14]. Procedural generation has been of wide interest because of its power of generating at run-time without the need of storing anything. It also encourages extendibility and eases variation in objects. An example use of procedurally generated environment is Procedural City Generation by Shamus Young [You09], which uses no art assets. A more general approach to procedurally generating complex 3D shapes, Paul Merell and Dinesh Manocha has written a paper on modelling algorithms, synthesis, algebraic constrains and boolean expressions [MM10].

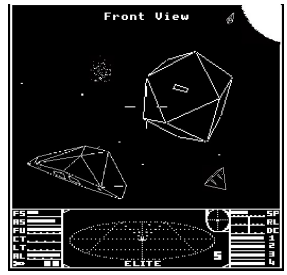



**Figure 2.1:** Elite game from 1984.

## 2.3 Augmented Reality

We wanted the procedural generation to happen along with Augmented Reality (section 3.1). We found that something similar has already been made by Qualcomm Vuforia[5], but focusing much more on AR, rather than procedural generation [Qua13]. We also wanted to know the posibility and extendibility

[4] World of Level Design, http://www.worldofleveldesign.com/
[5] Qualcomm Vuforia, http://www.qualcomm.com/solutions/augmented-reality

of identifying markers [AfARA05] [Mac98] [RD12]. The last cited is the closest to our vision. Later, we decided not to focus on identification of augmenting markers, but more on procedural generation. For this reason, we used a Unity integrated package from Vuforia [Qua12].

## 2.4 Related work

Moving closer to our subject in mind, we found a page [Dou14], which covers a wide range of Procedural Content Generation (PCG). In it we found references to PCG games, software, algorithms, events, articles and much more. An infinite procedurally generated terrain by Peter Jones is done using relative random heights with similarities to Perlin-noise [Jon13]. A 3D artist with expertise in games programming is introducing to procedural geometry done in Unity [Sur13]. Unity has even got a documentation in their manual, explaining the generation of mesh procedurally in Unity [Uni14]. A real-time application of terrain generation, done by Jacob Olsen, shows an example as early as 2004, using erosion and height-maps [Ols04]. For hydrology along with terrain, a paper from LIRIS, Université de Lyon, presents how terrain generation is done using simulated water and rivers [JDG13].

On top of terrain generation, we generate roads. Example of procedural road generations is done by SixTimesNothing[6] and another example done by Martin Glaude [Gla13]. A belgian CG-artist Kim Goossens has done procedural generation of roads, very similar to what we want to achieve, but with different approach [Goo13]. He uses Houdini as platform, and does procedural generation by manually changing control points.

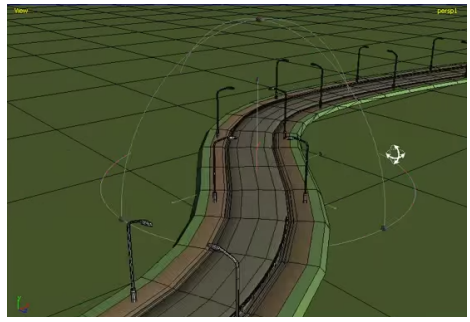


**Figure 2.2:** Houdini implementation of procedural roads by Kim Goossens.

[6] SixTimesNothing, http://www.sixtimesnothing.com/

Chapter 3

# Design and Tools

In this chapter, we will focus on the different methods used to solve the particular problems defined in the problem statement section. These methods will include analysis of the given problem of its own. We will also discuss other alternatives to our design choice. Once the method to solving the problem is established, we will try and implement it in the next chapter.

## 3.1 Creative Process

As the objective of this project is to make an invention, we had to make use of different techniques in order to define and test our concept.

### 3.1.1 Brainstorming

The initial project definition process for us was a simple brainstorming. The subject was gaming, and how we can contribute to that world. Our goal was to make a game, as we have a passion for the field. However the main criteria was that we needed to create something new, not in the game concept, but in the technical aspect.

Our idea lead us to Augmented Reality (AR), which we believed had potential and unexplored possibilities. We found, what we believe was a fun idea for a game. A treasure hunt you could make in your own house, with geographical locations (GPS) as AR markers. After doing some research in the field of AR and GPS, we found out that it is still an imprecise technology (without standard AR markers), unless with predefined locations and a lot of image analysis calculations. This took away the liberty and interactability that we wanted, and we decided to make some sort of a scavenger hunt. However there are already many AR scavenger hunt games [Cha14].

Keeping in mind AR, we thought of brainstorming on the field of level design. Not being experts in the field, we could only come with our superficial impressions of it, and state that we thought the process could be optimized. The idea of being able to visualize and edit levels using AR arised. We thought of it being an interesting concept to visualize levels one had created on paper. We believed this could help level designers share their ideas with team mates, and perhaps help them in their creative process.

### 3.1.2 Sparring with supervisor

In the process of defining the concept and project, we got a lot of help and tips from our supervisor. Sparring with someone who can see the process from outside, has proven helpful many times. It has not only ensured that the concept would orient in the right way, but also that energy and focus was spent on the right areas. He also came with ideas and suggestions, that helped us think out of the box.

### 3.1.3 Interviews

After having a clear idea about making a level design tool and what we wanted to do achieve, we decided to test out the concept on actual level designers. We had a rough prototype showing the concept, and triggering the imagination and interest of the level designers. We contacted many level designers from different origins, and interviewed them.

We had prepared questions (can be found in the appendix A), where we would ask them about level design, for us to understand a lot more about how they think and what they could need. After we had finished our interviews, and reviewing all of them (Appendix A), we realised that visualizing levels was not an issue, and therefore level designers would not be interested in using the

program we would develop. We also found out that the AR was just a gimmick and that it had no real interest for the project.

### 3.1.4 Thinking out of the box

Having realised that level designers would not need our project, we were forced to rethink our project. We got the idea of changing the target group. In fact one level designer suggested, that what we wanted to develop, was a much better game concept than tool. Therefore we found ourselves making a tool which would allow anyone to easily make levels, by allowing users to draw on paper, scan those in a mobile device and play with their level.

As a matter of fact, the aspect of being able to make a game your own, is something which is appealing to a lot of people. Moreover some successful games have arised from user created games, using level editors published by bigger games. An example of this is **League of Legends**, which started off as a custom map called **Defense of the Ancients** in **Warcraft 3** [Ngu12]. Not to forget one of the greatest "make your own level" games, **Minecraft** [Moj09], which allows people to create universes from building blocks.

## 3.2 Unity3D

Our main tool for testing our procedurally generated environment is the Unity3D game engine, but *why* Unity? It is free and offers an interactive environment to visually run and test on. Unity also optimizes meshes, and generates materials and shaders. Additionally, we can separate meshes, scale and move them at will while debugging. Unity's scripting development environment, MonoDevelop, also has a powerful debugging tool, allowing run-time debugging with breakpoints and variables inspection, by either use of the editor inspector or MonoDevelop variables watch (Figure 3.1). See Figure 3.2 for the editor view in Unity.

Other alternatives we could have used for our development would be developing within the Visual Studio environment or other game engine editors such as UDK and CryEngine[1]. Had we used Visual Studio, we would probably have to use OpenGL, but then we had to operate almost directly with the CPU in order generate the meshes most optimally. We would also have to create the run-time environment, camera and player movement, initializing buffers and

[1] CryEngine, http://cryengine.com/

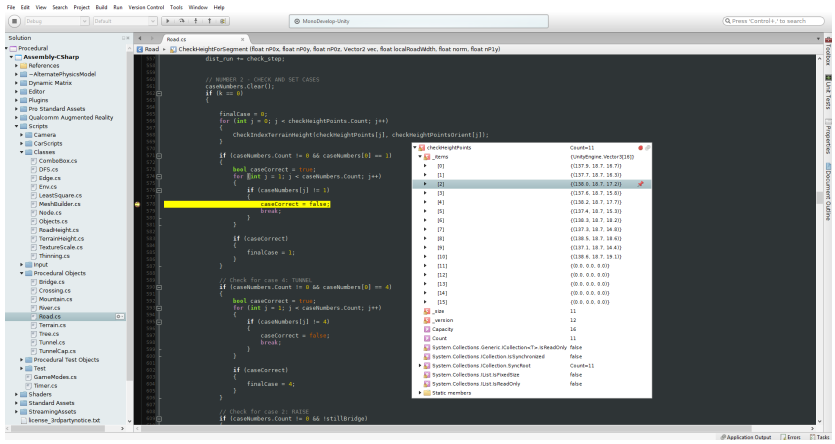


**Figure 3.1:** Debugging in MonoDevelop. Debugged variables can be seen while using breakpoint.

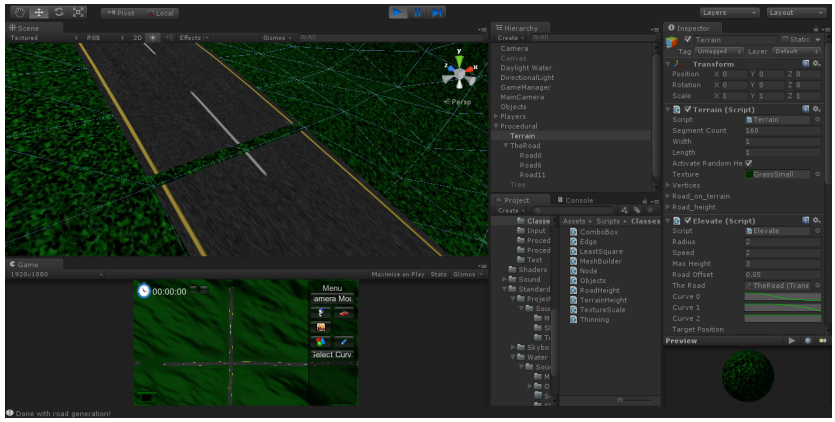


**Figure 3.2:** Editor view in Unity. Inspector window debugging can be seen on the panel on right side while also debugging the mesh on the scene view.

shaders. Using Unity therefore, saved time for us and allowed us to focus on interesting problems.

UDK and CryEngine are also alternatives to Unity. Both of then can be downloaded with educational use, but the main reason we went with Unity was our past experience with the tool. Therefore no time was wasted getting acquainted

with new software. Our past project [Jø12] was also in Unity, recommended by our previous supervisor. This also made it easy adaptable and reusable for this project.

In scripts, we set the variables we want to debug as either `SerializeFields`[2] or public variables. One of the powerful things with this, is that it can be edited run-time as well. This enables further dynamic testing, without the needs to execute the generation over and over every time.

## 3.3 Game

In order to test the levels we have created, we created a simple racing game, which allows you to traverse your scene. See Figure 3.3.

We had considered many different genres in the beginning of the project. For instance a physics action game. The idea with roads would then be path-ways for the character to walk on, and different obstacles in the field would be interactable. When we talked to level designers (Appendix A), they mentioned assets like push boxes, shooting balls, events, puzzles etc. For that, we tested a game where you have a third-person character control. Another idea was to make a shooter game, where the player shoots at obstacles in the generated environment or even make a playground area for multiplayer environment.

But the one that fit the most was the racing game. For the game, we simply needed a car, first-person or third-person, and drive on roads with hills, fences, bridges etc.

## 3.4 Curve fitting

This mathematical problem consists of fitting a curve to a set of points. We will extract control points which define the curve as a cubic bézier.

[2] http://docs.unity3d.com/ScriptReference/SerializeField.html

**Figure 3.3:** Car game.

### 3.4.1 Least square

The theory behind least square is taking the distance between the points on the curve and a mean, which is found by taking the average of every point's x and y. We square the distance from each point to the mean. We use these to determine the influence of that specific point, and compare them with the other points. Figure 3.4 illustrates this.

The first graph shows the points of the curve and each point is defined as a 2D vector

$$P = [p_1, p_2, ..., p_n] \quad , \quad p_i = \{x, y\}$$

The means are found as shown on the second graph, top-right. On the last two graphs, we find the distances from each point to the mean on both, x and y values, respectively. These are then used to find the linear line equation

$$y = x_0 + cx = x_0 + \frac{\sum(x - \bar{x})(y - \bar{y})}{\sum(x - \bar{x})^2} \cdot x$$

where $x_0$ is the intersection with the y-axis and $\bar{x}$ and $\bar{y}$ denoting the means of the points in x and y.

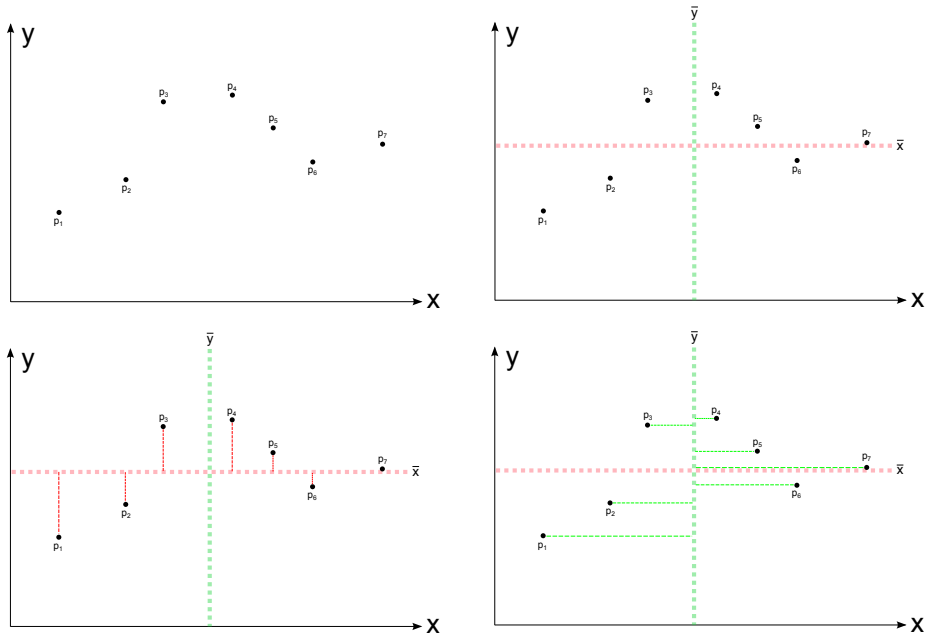



**Figure 3.4:** Least square distance from points.

### 3.4.2 Bézier fit

Seeing as we do cubic bezier, we are working with polynomials of third degree. We therefore need to calculate least square in a different way. This is done by a linear combination of equations and matrices. Note that the following equations are derived from Jim Herold's article [Her12].

We use a cubic bézier function here, which means that our curves can only bend two times. The reason is that only four control points are used and two of them can make the curve bend. This is a design restriction which we need to take into consideration, when generating the road paths.

We have the formal bézier equation given as

$$y = c_1(1-t)^3 + 3c_2t(1-t)^2 + 3c_3t^2(1-t) + c_4t^3 \tag{3.1}$$

We can split this equation into matrix multiplication of 3 matrices so they form the same linear combination as the bézier equation.

$$T = [t^3, t^2, t, 1], \quad M = \begin{bmatrix} 1 & 3 & -3 & 0 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \quad C = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix}$$

Multiplying these matrices together, we will get the same equation as 3.1. The next step is to give each point a corresponding t value within its bézier function. For instance, for $t = 0$ we have the start point $p_0$ and for $t = 0.1$ we have $p_1$ assuming we have 10 points, $P = [p_1, p_2, ..., p_{10}]$.

To find the corresponding $t_i$'th value of the points, we have to get the normalized path lengths by first taking the path lengths of each segment and then divide with the sum of the lengths

$$t_i = \frac{|p_i - p_{i-1}|}{\sum_{i=2}^{n} |p_i - p_{i-1}|} \tag{3.2}$$

In the previous example with linear least square problem, we used the approximation squared distances from the mean. For the bézier curve fit approximation, we are using a similar one, but with matrices, namely the residual sum of squares. As previous, we take the x and y values of the points separately.

$$E(c_y) = \sum_{i=1}^{n} (y_i - B(t_i))^2$$

where we try to find the control point's y-value, $C_y$, $E()$ is the error and $B()$ the bézier function. To find the maximum likelyhood estimation, we can rewrite the equation so that we maximize the probability of distributing all the points of the curve. To do that, we use another matrix, namely

$$S = \begin{bmatrix} t_1^3 & t_1^2 & t_1 & 1 \\ t_2^3 & t_2^2 & t_2 & 1 \\ ... & & & \\ t_n^3 & t_n^2 & t_n & 1 \end{bmatrix}$$

and then we rewrite the error equation as

$$E(c_y) = (Y - SMC_y)^T (Y - SMC_y)$$

where Y is the vector with the y-values of the points. Differentiating this function and setting the left-hand side to 0 will give us the maxima,

$$\begin{aligned} &\frac{\delta E}{\delta C} = -2S^T(Y - SMC_y) = 0 \quad \Updownarrow \\ &S^TY = S^TSMC_y \quad \Updownarrow \\ &C_y = M^{-1}(S^TS)^{-1}S^TY \end{aligned}$$

which will finally yield us the control point. The same equation applies for the x-component of the control point.

## 3.5 Binary Extraction

Since the input management is done via the camera of a given device, we need to translate the image of each frame (or a single frame) into readable data in our tool. This is done by using simple color distinction of pixels and performing image analysis algorithms to make the input usable.

### 3.5.1 Overview

The user scenario is that he takes an image of his level drawn on paper. This one contains black and blue curves. The black color indicates roads and the blue water. The goal in binary extraction is to retrieve data sets containing the points where we have identified roads and rivers.

As binary handling we have many different kinds of operations, all explained below. However we use them in a specific order, for us to get the result we want.

On figure 3.5 we see the process from generating the binary data, to sending nodes over to the DFS class (explained in section 3.6) for us to generate the environment (all processes will be discussed in detail in their subsections below).

The first thing we do, which is common to both rivers and roads handling, is extract a binary data from the image we scan. We identify two arrays, one of which containing 0s and 1s for the road, and the other one for the rivers. When there is a 1 in the road binary array, it means we identified a black color. Opposite if we identify a blue color, we will put in a 1 in the binary data of the river. Otherwise the binary data of both is filled with 0s.

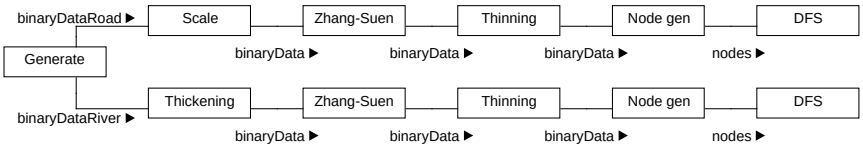



**Figure 3.5:** Overview class diagram of the implementation.

#### 3.5.1.1 Road

When working with roads, we will take the binary data which identified the black pixels. The first process we will do is scaling the binary data. This scaling will give us more space to work with when looking at crossings (section 3.13).

#### 3.5.1.2 River

When working with rivers, we will take the binary data which contains the identified blue pixels. Instead of scaling the binary data, we simply thicken it (reason explained further down).

#### 3.5.1.3 Common

After the thickening or scaling, we thin the binary data using the zhang-suen thinning, which we use because of its performance. However the zhang-suen algorithm doesn't thin all roads to only have one neighbour, so we complete it with a skeletonizing algorithm that we simply call thinning. This one ensures that the all the curves have width 1. When we have done all these we send the resulting binary array to a node generator, which will create the nodes we need to perform our DFS. In the end we send all these to the DFS class, which will generate the final road.

### 3.5.2 Color identification

The basic idea is that we draw black and blue curves on white paper. These are meant to be roads and rivers respectively. See figure 3.6.

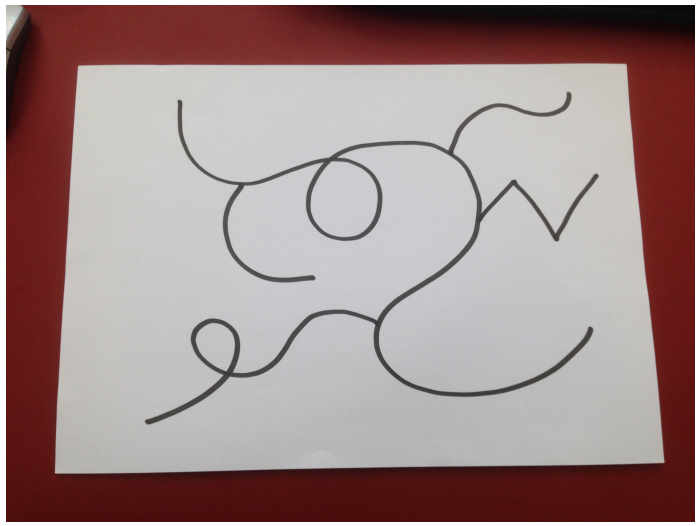


**Figure 3.6:** White sheet of paper with road

We read the image as a set of pixels. We assume that the background is white, and we will presumably only get either black, blue or white pixels. This is what we take advantage of: Converting color data into binary data.

### 3.5.3 Thinning

In image analysis, more specifically in Morphology, thinning is an algorithm that takes data, and iteratively forms a skeletonization by removing unnecessary pixels [RFW03].

Thinning uses two different filters to find and remove unnecessary pixels. These filters are used when checking the specific pixel's neighbours. The filters are shown on Figure 3.8.

Figure 3.8 shows how we decide which pixel to turn into white. A 0 indicates the occurance of a white pixel, a 1 indicates the occurance of a black (or blue

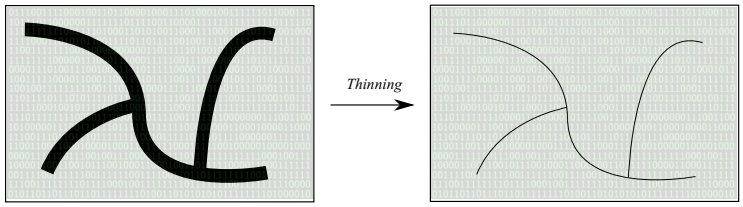



**Figure 3.7:** Thinning of road.

| | | |
|---|---|---|
| 0 | 0 | 0 |
| | 1 | |
| 1 | 1 | 1 |

| | | |
|---|---|---|
| | 0 | 0 |
| 1 | 1 | 0 |
| | 1 | |

**Figure 3.8:** Two types of filter matrices.

when working with rivers) pixel, and a blank means that we do not check that neighbour.

These filters are currently one-way oriented. To get the thinning algorithm to work properly, each of these filters need to be rotated 90 degrees and used on every pixel once again.

### 3.5.4 Zhang-Suen thinning

Similar to the previous thinning algorithm, this version has another way of checking whether or not a black pixel should be turned into white. This time, instead of using simple filter check on a pixel and its given neighbours, we check for transitions from white to black pixels [Rez13].

First, we start by giving an order to the neighbours around the pixel we are looking at. We call that pixel $P_0$.

| | | |
|---|---|---|
| $P_8$ | $P_1$ | $P_2$ |
| $P_7$ | $P_0$ | $P_3$ |
| $P_6$ | $P_5$ | $P_4$ |

**Figure 3.9:** Neighbours of pixel $P_0$

Next, we define two quantities

- First one telling us how many pixels in the neighbours are turning from white to black in the sequence $P_1$,$P_2$,...,$P_8$. For instance, if pixel $P_3$ is white and $P_4$ is black, then the number of transitions is incremented. Thereby, the number of transition is between $T \in [0; 8]$. We will call this quantity $A$.
- Second one will simply give us the number of black pixels around pixel $P_0$. So we have $\sum_{n=0}^{8} P_n$. We will call this quantity $B$.

And finally, we must ensure that all of the following conditions are met before we can decide whether or not to change the pixel to white:

**Step 1:**

- **Condition 1:** B is between the values 2 and 8.
  $2 <= B <= 6$
- **Condition 2:** A is black.
  $P_0 = 1$
- **Condition 3:** One of the following pixels is white.
  $P_1 = 0 \vee P_3 = 0 \vee P_5 = 0$
- **Condition 4:** One of the following pixels is white.
  $P_3 = 0 \vee P_5 = 0 \vee P_7 = 0$

**Step 2:**

- **Condition 1:** B is between the values 2 and 8.
  $2 <= B <= 6$
- **Condition 2:** A is black.
  $P_0 = 1$
- **Condition 3:** One of the following pixels is white.
  $P_1 = 0 \vee P_3 = 0 \vee P_7 = 0$
- **Condition 4:** One of the following pixels is white.
  $P_1 = 0 \vee P_5 = 0 \vee P_7 = 0$

Notice that we check two steps for each pixels. When all these conditions are met for the pixel, the checked pixel will be removed (set to a white pixel).

### 3.5.5 Thinning and road widths

As we will explain in 3.9, we were interested in introducing road widths from user input. This means that we wish to have the road widths drawn on the paper, to be transfered into our program.

This, we found, fixed a lot of our problems, in input consideration. If we have a fixed road width, we can risk that neighbouring roads can cover each other in a level, but not on the paper. By allowing the user to define the road width of every single segment, roads will not cover neighbouring roads, as they will not on paper. See Figure 3.10.

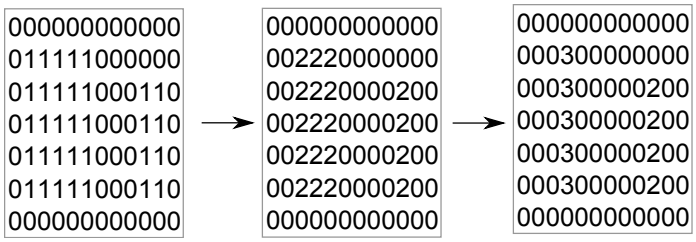



**Figure 3.10:** Thinning in steps. As seen, the number gets incremented for each iteration of thinning.

We found a very simple way to do this. Every black pixel subjected to thinning, will be affected at a certain iteration of the Zhang-Suen or skeletonizing algorithm, and this iteration will define the road width. A road with width value 5, will be thinned 2 times (two on each side), and the resulting node (section 3.5.6), will then be able to remember this value and send this information to the roads. Of course we will not take into account the first iterations, when defining the road width, as those only happen because of scaling.

### 3.5.6 Nodes and Edges

In order to use the binary data, we needed to create relations between the black pixels. We needed to give each black pixel a unique identifier, and a set of neighbours it is connected to.

We decided to make each black pixel into a node, and create edges between the black pixels that are neighbouring it. This means if a black pixel is next to another black pixel, we will create an edge between them.

Regarding their identifier, and their coordinates we decided to extract that straight from the image. Their coordinates are taken from the pixels image position. Hence a pixel located on the 5th line and 8th row, will have coordinates x = 5 and y = 8.

### 3.5.7 Scaling

After having checked some special input cases, and difficult situations to handle, we found a solution that would help in all cases. In fact many of our problems happen at intersections, and when these intersections are too close to each other. The solution was therefore scaling the image we receive, so that one pixel becomes 4 times as big, which means one pixel becomes 16. After thinning the whole image we get the same image as we would without scaling, however with more steps between intersections, which simplifies implementation a lot (this will be explained further in section 3.6 and 3.13).

Of course there is a performance drop because of this, because we have to do 4 times as many calculations. However the amount of errors dropped and we get more credible solutions. See section 4.3

### 3.5.8 Thickening

When drawing blue over black, the outcome is black. Therefore when generating binary data for both river and road, we will have holes in our river binary data due to black being dominant over blue. We decided to implement a thickening of the binary data, which simply turns the neighbours of a 1 to 1. See Figure 4.5.

There were many different options to explore, namely we could decide that when identifying a black pixel, we also identify a blue pixel. Then however we would have to check if there is blue pixels around, for else there would be rivers under every single road. This seemed complicated, so we thought of thickening.

The advantage of thickening the rivers is that we have to do no specific checks. Given any two points, thickening them enough, will make one connected area. Therefore the issue at hand, is how much to thicken, so components that are meant to be connected get connected, and those who don't will not. The solution we found was to take the maximum road width (which we get from thinning the road), and use this as a thickening factor. This means, if there is a river around the thickest road it will get connected.

The downside of this, is the fact that componenents that shouldn't get connected could wind up getting connected. With very thick roads, and rivers that are very close to each other, we could end up with having one big river, instead of many small ones.

Once we have all the connected components we want, thinning them will give nice rivers that will look as if we were able to scan it despite the black lines. See Figure 3.21.

## 3.6 Depth-First Search

In order to go from nodes and edges to roads, we need to make logical relations between neighbouring nodes, add those together to generate segments of roads and rivers, and create relations between these.

### 3.6.1 Choice of algorithm

After having generated nodes and egdes as explained in 3.5, we decided to run a "Depth-First Search" algorithm (DFS).

The reason we went for the DFS algorithm, is due to its very intuitive implementation in the context we need it for. It traverses a path all the way, before looking at the next one. Therefore the DFS will only jump back, when it has reached the last node of a path. Hence we can flush the points we have so far to the road we are working on, jump back to the previous node(s) and start a new road.

An alternative could be to use a Breadth-First Search. This one will expand through all the network before jumping back. Therefore much care had to be taken in the implementation, as to differentiating one road from another, and which road a specific point is contained in.

### 3.6.2 Nodes and Edges

As explained in section 3.5 we created nodes and edges for our DFS to search on.

The **nodes** are information holders. They indicate a position, id, roadwidth (purpose explained in section 3.9.4), neighbouring nodes and edges.

The **edges** are used to connect one node to another. The edge knows which two nodes it connects.

Our DFS traverses every edge in the graph, and once an edge is used, it is deleted. The idea is going through every edge once, hence every node will get visited. The nodes however can be traversed several times, as we have crossings and circles, where you need to go through a node again.

### 3.6.3 Segments

In order to decide where we start our DFS search, we choose the node which has only one neighbour. In fact we found that one neighbour indicates an end of a segment. So when we wish to start our search, we prefer to have it start in these points, as we know it will not return.

After finding nodes that are on the same segment, we have some restrictions about these segments. The first regulation is about the minimum amount of points accepted to form a segment. Least square only works with a minimum of 4 points, and therefore one restriction could be to have a minimum of 4 points in a segment. However we see 3 alternative cases:

- 1 point: We decided to throw away segments of size 1. We justify it as in the implementation of roads 3.9 we need an orientation vector, and such one cannot be found with only one point
- 2 points: We have now a start and end of a segment, and therefore we need not least square to make calculations, as little has to be done to make it work for a road
- 3 points: Like before we have a start, end but this time we add a middle point.

Alternatively one can argue on whether there should be a limit as to how many points there can be in a segment. In fact we found that cubic bézier can only make two swings at a time, and therefore a restriction is needed. We had some considerations. One was to count the swings in the DFS implementation. This would be optimized as a minimum amount of passes would be needed, however the implementation was too complicated for the optimization granted.

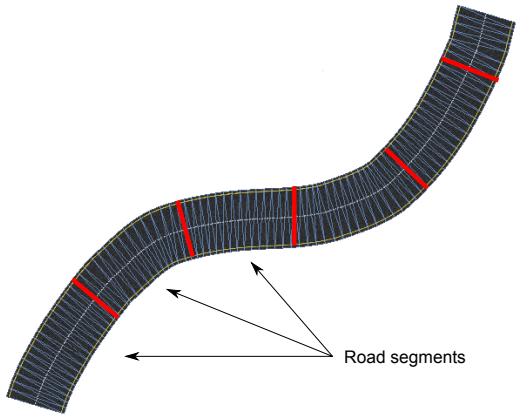



**Figure 3.11:** Road segmentation.

We decided to have a maximum amount of points per segment, and when that value was exceeded we would flush, and create a new segment, of the same road. Furthermore we found out that counting the amount of points in a segment, and passing that to the road, so it would not make too many vertices, gave a clear and optimized result everytime.

Knowing the segments, and knowing which roads they were segments of, we had to find a solution regarding how to make them connected. To do this we have a lot of data structures that keep track of this which is explained in detail in section 4.4. There were a few options.

- Sending the two last vertices of the previous segment to the next
- Building the mesh dynamically as we traverse the road
- Adding vertex points to a list, and building it all together in the end

We chose the latter one, due to conveniance in the road implementation which will be discussed in section 4.9.

### 3.6.4 Crossings

Having thinning in our program, it was quite easy to identify crossings: a node with more than two neighbours is contained in a crossing. We therefore had

to make a new type of search: which nodes are contained in a crossing. We decided to make a separate DFS. The reason was two-fold. First of all we are interested in exploring the depth of the crossing before continuing, and second of all it cohered with the implementation of the road DFS.

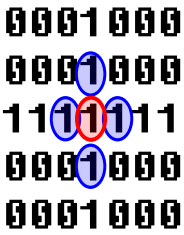



**Figure 3.12:** Crossing identification. When we reach the red point, we check the neighbours (blue points).

We therefore had to add a depth (decreasing for every step) to the DFS so it did not run through the whole data structure, but stops when a certain depth was reached. In this way we end up with a set of nodes that constitute the crossing. As we use the DFS, nodes from the same side of the crossing, will be one after the other.
An option could have been to use a Breadth First Search, however we would have had more work in matching nodes of the same side should we need them to create segments (discussed in section 3.13). However for depth implementation we would have had less issues.

## 3.7 Terrain

The terrain is the ground from what all our environment objects is built. It can represent anything from sand to grass or even water. The terrain is in its most simple form a planar quad with 4 vertices and a face (2 faces for triangulation).

### 3.7.1 Mesh

The terrain has a number of vertices and faces, starting out laying flat in a 2D plain, but can also have its third dimension representing the height and creation of hills (explained in section 3.7.2). The more vertices and faces the

mesh consists of, the better resolution it will have when creating a huge scene with a big environment.

The creation is done by using a triangle-strip going through each vertex.

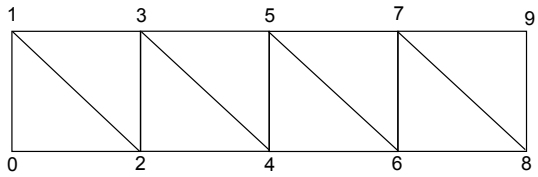



**Figure 3.13:** Triangle strip from vertex 0 to 9.

### 3.7.2 Height

The terrain in our environment is the most simple but yet can turn into a complicated procedural mesh. The reason being its varying heights across the environments. For that, an algorithm called Perlin-noise [Jø12] is used to create random, but structured curves along the surface of the terrain. To make the terrain not look so smooth and unnatural, an additional algorithm called turbulence is used to give the surface of the smoothed terrain a disturbing surface. This can for instance be used to resemble rocky hills. (explained in [Jø12])

However, whenever there is a Road 3.9 or River 3.10, we want the terrain heights to act in a different way.

#### 3.7.2.1 Road heights

We have two situations for the road to behave

1. The road follows the *same* height function as the terrain.
2. The road has its own height function *independant* from the terrain.

In situation 1, we do not have to worry much about many cases as the road follows the terrain. The only worry is that the road is more smoothly layed

on the ground than the terrain ground. So, a way to visualize this in the most meaningful way, we generate the terrain as having noise and turbulence whereas the road only makes use of noise.

By doing this, we ensure that we have smooth hilled roads, that you can drive on. To also smooth the edges outside of the road, we check, stepwise, a distance further away from the road width.

In situation 2, it is not as trivial. In this case, we have to think about further 4 cases in which the road can be. These can be seen on figure 3.14.

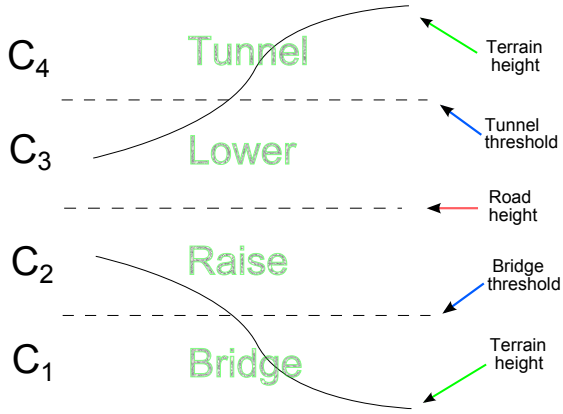



**Figure 3.14:** The four terrain height cases

When the road height is below a certain threshold, a bridge is created underneath. Above a certain threshold, tunnels are created and we create a "hole" through the terrain. In other cases, we either raise or lower the terrain height to fit the road height.

## 3.8 Bézier

A bézier curve is a parametric curve which consist of a set of control points. The curve uses these control points to align its smooth curvature by any specific weights on them. There are linear, quadratic and cubic bézier curves and the most generalized bézier formula, namely

$$B(t) = \sum_{i=0}^{n} \binom{n}{1} (1-t)^{n-i} t^i P_i$$

$$= (1-t)^n P_0 + \binom{n}{1}(1-t)^{n-1} t P_1 + ...$$

$$... + \binom{n}{n-1}(1-t)t^{n-1} P_{n-1} + t^n P_n, t \in [0,1]$$

For our project, as we work with 3D environment, we chose to use the cubic bézier curve, which in this case would result in the formula

$$B(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t)t^2 P_2 + t^3 P_3, t \in [0,1]$$

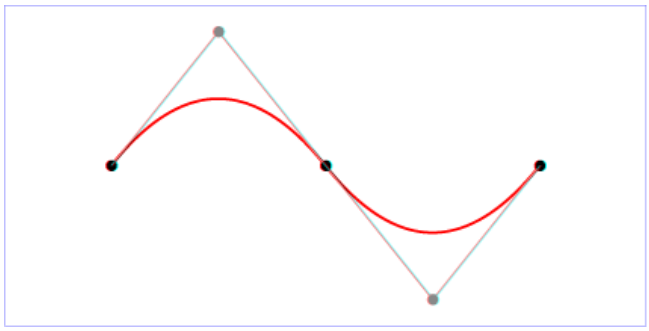


**Figure 3.15:** Example of a cubic bézier curve.

## 3.9 Road

Our program's main feature is generating nice looking roads. It's important they are coherent to the input, and smooth. At this stage we have already read and translated the paper input, and the question is what to do with this data.

### 3.9.1 Road mesh

We decided to implement the road as a simple triangle strip (Figure 3.13). It seemed as the simplest and best solution, as it is exactly what we had done on our bachelor thesis [Jø12]. The idea is taking the orientation of one control point to the next, and use its hat vector (the one which is perpendicular) to give width to the road.

### 3.9.2 Aspect of road

We had few considerations as to what information to give the road and whether we should use Bézier 3.8 or raw data from the binary.

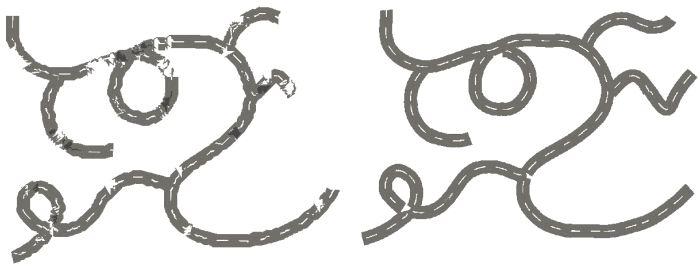


**Figure 3.16:** Raw vs smooth roads.

The problem with using the raw data, as shown above, is the fact that roads will be very edgy, and more liable to data errors. On the other hand, we would not have problems with partitioning roads into segments and making them fit (see Figure 3.11).

In fact one of the big challenges with using Bézier, and Least Square approximation, is the fact that we need to restrict the amount of data given to the curve fitting, as it cannot create roads with more than two turns, and we need to partition roads into a set of segments. One challenge is therefore to make them connect and fit, and optimally make the transition between one segment and another invisible.

The first step is not having gaps in between them. Without gaps the road looks like one, and not like a set of smaller meshes. The next step is making the first two vertices of the next segment, fit with the two last vertices of the previous segment.

The solution we chose was to do two things:

- Transform the first control point of the next segment, to the same as the last control point as the previous segment
- Make a vector interpolation for the first few bezier steps between the last vector of the previous segment, and the current calculated vector

### 3.9.3 Random height

We decided that random height would remove a lot of uniformity, and make environments generated a lot more interesting, and fun to play with.

To start with, we needed to decide on reality of road heights. In fact one could have tilting roads when the road is turning to one side or the other. However we decided that the effort we had to put into it, compared to the benefit it would give the levels, was too little. After looking at real-life roads, most of them do not tilt according to turn, unless it being a race track. Also it would add a lot of complications for us in setting the height of the terrain. In fact we would need to interpolate the height of the terrain according to the tilting angle of the road.

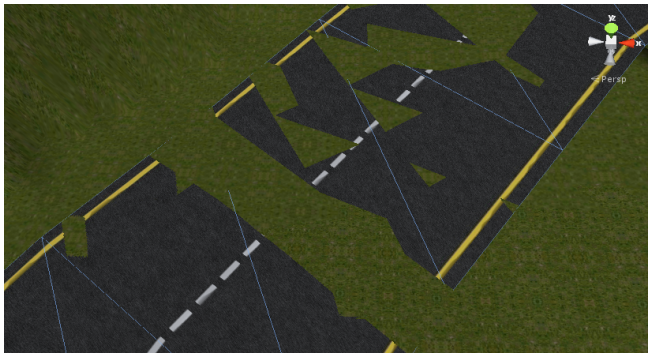



**Figure 3.17:** Error with some terrain vertices being over road height.

This simplified the implementation for us, as it would mean each vertex pair would have the same y-value, and all we would have to worry about is the height transition between one pair of road vertices and the next. However using the noise function, we ensure that the next value is close to the previous, and that "an understandable" road-height system appears.

### 3.9.4 Road widths

As we do thinning on the binary data extracted from the scan of the paper, we lose the road widths, the user could have given his roads. However as explained in section 3.5 we can get a good estimation of the road width at one specific point. With those raw values, just as explained previously about the aspect

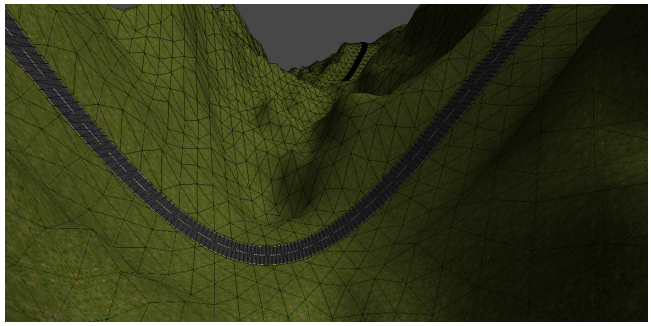


**Figure 3.18:** Road following the terrain height.

of roads, we decided to make interpolations between these values, in order to minimize mistakes. To start with, we decided to give each segment a road width, as each segment would not cover too much distance. This road width would be found from taking the average of the raw values.

The way we decided to use these road widths, is by interpolating between the previous road width and this road width in the first half of the segment, and interpolating between this road width and the next road width in the second half of the segment. This way we have a lot of smoothing between road widths, and small inconsistencies will get corrected.

## 3.10 Rivers

We decided to introduce rivers to the program as to allow the user to make levels more interesting.

We decided to make rivers in a very similar way to roads. In fact the system of reading black pixels and translating it to spline coordinates, is re-used for rivers. This time however we will not create a mesh for the river, but affect the terrain. We have set a 0 level, where there is water, and the rivers will appear when the terrain is lowered to under 0. The challenge therefore lies in making sure we smooth a transition from normal terrain height to under the 0 height level (as opposed to what's explained in section 3.7).

## 3.11 Tunnels

Tunnels is a way to encapsulate areas in the terrain, where there is a hill or mountain above or it could be a hole dug into the underground. In any way, tunnels are created to bypass certain areas in the terrain. Because of that, it is highly dependant on the formation of the road.

A tunnel is an arc above the road, created by the 2D vector trigonometry, i.e.

$$\vec{v} = \begin{bmatrix} c \cdot cos(\theta) \\ c \cdot sin(\theta) \end{bmatrix}$$

where c is the radius of the tunnel. As we go from angles $0^o$ to $90^o$, in $15^o$ steps, we would have the half tunnel looking like the following illustration

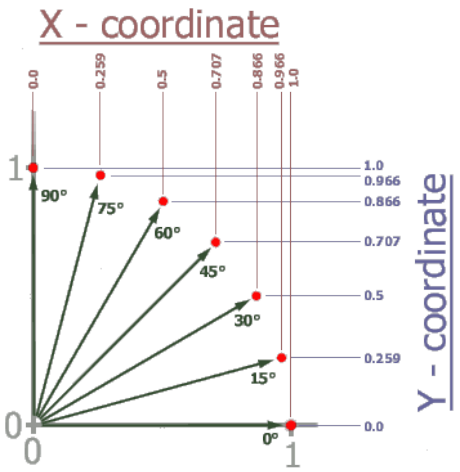



**Figure 3.19:** Half tunnel. To have a full tunnel, we turn 180 degrees.

### 3.11.1 Entrance

The tunnels themselves do not make much sense without an *entrance*. The entrance will, not only emphasize the tunnel, but also connect it with the surroundings.

### 3.11.2 Hole

When identifying case 4 as explained in Figure 3.14, we wish to create tunnels. However leaving the terrain unaffected will create the problem as shown on Figure 3.20.

There were many considerations regarding the design of the tunnel going through a surface (hill). One consideration was to make a depth mask shader on a mesh on the entrance of the tunnel so that would ignore the rendering of the surface mesh and only render the relevant meshes. The problem here was the small gap between the tunnel entrance and the vertices covering the terrain.

So we decided to make the terrain be lowered instead, and make a new mesh, representing to be the cap of the hill. We will take the same set of vertices, from the tunnel entrance to tunnel exit, and take the terrain height from the hill normally, and connect the edge vertices to the tunnel entrance.

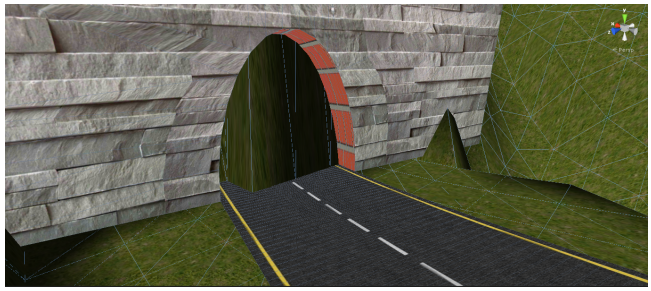


**Figure 3.20:** Terrain covering the tunnel entrance

## 3.12 Bridges

Having created rivers and gaps in the terrain, we decided to introduce bridges. The implementation of these is also based on the implementation of roads. They have to be under the road, and seem as an extension of these, however with a height, and an arch.

One possibility for our design, would be to make different kinds of bridges. The choice of the bridge would depend on the width and height of the road: a long bridge would not have the same type as a short bridge. However we did not estimate this as a crucial task, rather as a nice to have.

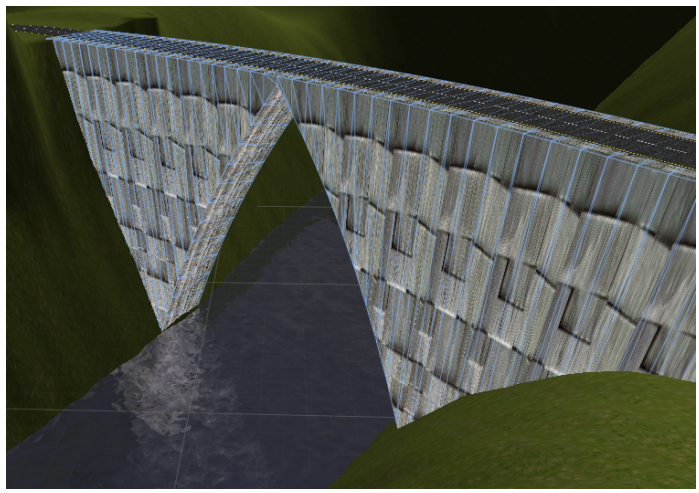


**Figure 3.21:** Road with bridge and river.

## 3.13 Crossings

Crossings rise an interesting problem in the program. In fact in real life, two crossings rarely look the same. This means we have some degree of freedom regarding their aspect.

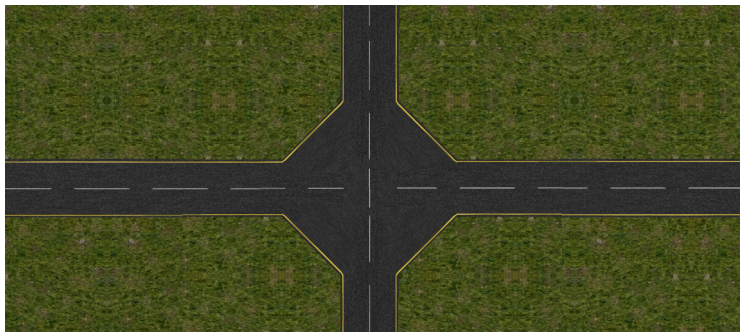


**Figure 3.22:** Crossing.

### 3.13.1 Choice of types

There are many ways of making crossings. The option we chose, is to make an extension of the road in question, and merge this one with all the other roads 3.22. This means the real challenge lies in making the transition between roads and crossings invisible. To do this we could simply pass two vertices per road, to the crossing and build from there.

Another option could be to decide on a dominant road and side road. This means that one road would try to join another road, but you cannot tell on the first road that there is a crossing. To do this, one should simply lower the height of the side road, in order for there to be a dominant one.

Building on this idea, we could have a standard crossing mesh, which could cover over the area where roads meet. This would be the simplest solution, however would not look convincing to the user.

Another consideration in that regard was that two dominant roads could not cross at all. To do this we could make a bridge where there was supposed to be a crossing so one road would go over another instead of crossing it.

### 3.13.2 Tilting

Regarding crossings, we had to consider what to do with its tilting. We decided to have no tilting on the crossing, no matter the tilting of the intersecting roads.

If we were to tilt it, it would work well for some roads meeting the crossing, but not all. Therefore it would affect the actual playing of the level. Another option would be to connect all roads end vertices together without adjusting their height. This would create twisted crossings, and a car would easily get stuck.

### 3.13.3 Bridge

As it is, the user can decide to make roads meet over a river or lake (even though it is very unrealistic). Therefore we had to take this into consideration, and make a bridge mesh where there is a crossing. Seeing as crossings group roads together, we assumed a cylinder shape was good. This one automatically

groups other bridges together, and does not need more work than necessary in order to make it look good.

### 3.13.4 Tunnel

We have more control over tunnels. In fact tunnels appear only where the difference in height between a road and the terrain is too big. However this is not user-defined, it happens randomly. In fact there is very few examples of tunnel crossings in real life, and we do not believe it adds much quality to the program. Therefore we lower the terrain around crossings, in order to make sure tunnel crossings do not happen.
However we also have the possibility to make these tunnel crossings. The idea is to make a crossing on the top of each tunnel that would fill the gap at the top. Then it's a matter of matching the right side of one tunnel to the left side of its neighbour.

### 3.13.5 3 main problems

We identified some problems with crossings that are close to each other, illustrated in the figure 3.23 below.

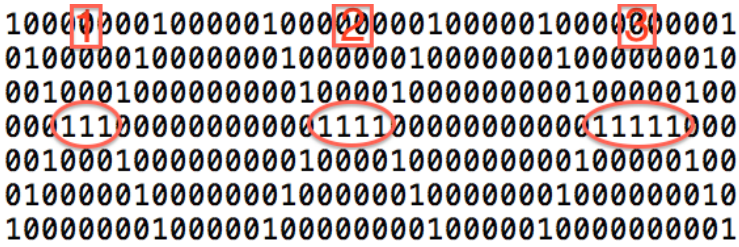



**Figure 3.23:** Crossing Issues.

In the figure we imagine roads meeting in crossings. In the different cases there is more or less distance between each crossing.

In case 1, we see the number 1 trapped in between two others, who have two neighbours, and therefore are crossings. In this case we want to make one big crossing, which covers the whole area.

In case 2, we have one more 1 between the crossings than in case 1. It is still not enough to make a segment between the two crossings, and making one big crossing would seem odd.

In case 3, we have one more 1 between the crossings than in case 2. We have now enough to make a tiny segment of one road step between the two crossings to connect them.

We realised that those solutions were not optimal, and that we needed more steps between them to get better results. Therefore we thought of making the scaling, which would turn one 1 to four 1s, and therefore give us more points to work with, and get a work around those problems everytime.

Chapter 4

# Implementation

In this chapter, we will explain how we implemented our design. We will focus on the structure of object oriented design, the platform we used to develop our product and more practicalities used in order to achieve our goals.

The chapters are structured so that there is a link between the design and implementation chapters.

## 4.1 Overview

We have chosen to implement our solution in Unity. Unity has many libraries, built-in shaders, defaults materials and utilities to run a basic game environment. Furthermore, it is much more flexible in its sense of generation and manipulation of meshes at run-time.

### 4.1.1 Class Diagram

We decided to make object orienting programming. The reason is the straight analogy between a class and an object in our program. For example, the road

class will do everything related to the road meshes, where the tunnel class will do everything related to the tunnel meshes. The most important classes are marked with red. See figure 4.1. As you can see, many of our classes

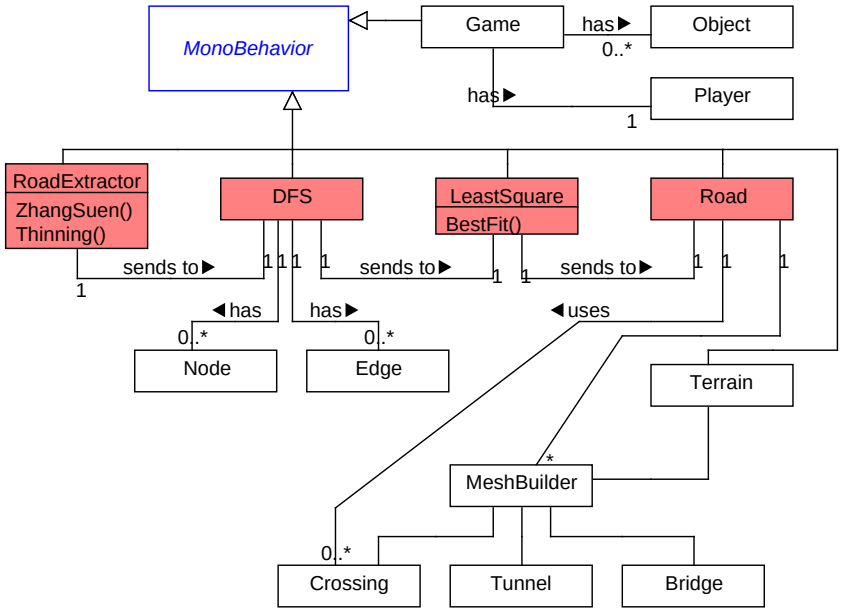



**Figure 4.1:** Overview class diagram of the implementation.

inherit from *MonoBehavior*, which is the main abstract class that defines how the Unity3D engine runs. The class `RoadExtractor` is the class responsible for taking a snapshot from the camera (see 4.3 for details) and uses the algorithms to thin the road with. `LeastSquare` is the curve fitting class and `Road` is the class responsible in creation of the road from the control points it gets from `LeastSquare`.

## 4.2 Least Square

The implementation of the least square algorithm requires matrix algebra setup. For that, we could either use a predefined matrix library or make our own multiplication and inverse functions. We chose the latter because it was not

hard to implement and we wanted to make sure that we did it right and had complete control.

To name them all, we have

- `MatrixInverse`
- `MatrixTranspose`
- `MatrixDeterminant`
- `MatrixMultiplication`
- `GetMinor`

`MatrixInverse`, `MatrixTranspose`, `MatrixDeterminant` and `MatrixMultiplication` are self explanatory. `GetMinor`, is a method to get the submatrix from a matrix. A submatrix is obtained by deleting a number of rows or columns in any given matrix.

In the implementation, we define two-dimensional array and instantiate the matrices defined in the design section. We create the normalized path lengths by taking the Pythagorean distance between each points and dividing each additional length with the number of points (as noted in 3.2)

Next we setup the matrix from all the lengths and the matrices composed by the x- and y-values, respectively. After this, it is a matter of multiplying all the matrices together, which result in our control points' respective x- and y-values.

The overall implementation of the least square algorithm is very similar to the formulas defined in the design section 3.4. The implementation can be furtherly checked on Appendix C, which is also very similar to the implementation from Jim Herold [Her12].

## 4.3 Binary Extraction

In Unity3D, the first thing we need to know is how to access a camera device's pixels. Then we need to edit and manipulate them through scripting (see Scripting API [API14b]).

We see that `WebcamTexture` along with `WebcamDevice` has a way of getting color pixels from the camera with a defined size. We chose to work with `160x120` because we believe it is sufficient. We could, in principle, work with larger resolution. It would affect performance and enhance precision of reading from the paper. We chose this resolution because we saw it suitable for real-time

generation.

The color pixels we get can be thought of as a matrix with each cell consisting of a vector of size 4 containing RBGA colors.

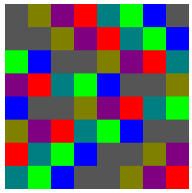


**Figure 4.2:** Color pixels. Each color consists of RGBA values.

### 4.3.1 Color identification

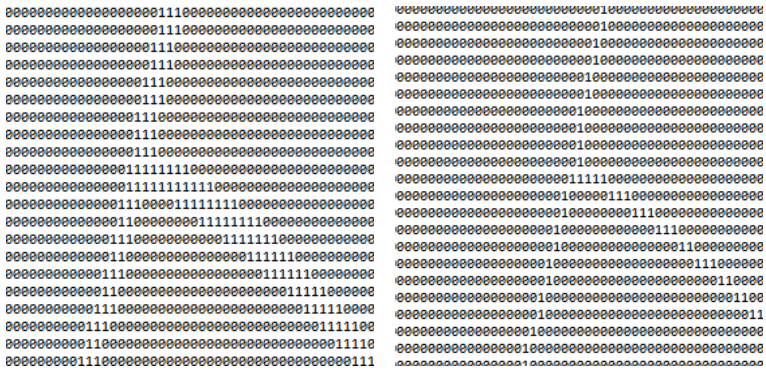
Most of the pixels that we take from the camera snapshot, will naturally be white with black lines, indicating the road. We assume that the camera is pointed right at the paper, with no background colors on the edges. If we decided to take the edges into consideration and focus on user experience, we would rather run into a detailed image analysis of the snapshot. As we had limited time, we decided to keep it simple.

From the readings of the pixels, we simply take their RGBA color values, and define them as black, if they exceed a certain threshold. In this way, we can easily get the values we want to manipulate with and omit the other color values. We set all the black color pixels to 1 and other ones to 0. Here is an example of such a reading (Figure 4.3a).

### 4.3.2 Scaling

The way we do scaling is by taking one spot and transforming it into a many. When scaling with a factor $f$, the size of the set will be scaled with a factor $f^2$. So in the example of a factor of 4, we get a binary data with a size which is 16 times as big.

When creating the scaled binary, we take the values from the small binary and insert them in the bigger binary as shown on Figure 4.4. We can setup the



**(a)** Thick road **(b)** Thin road

**Figure 4.3:** Thick and thin road.

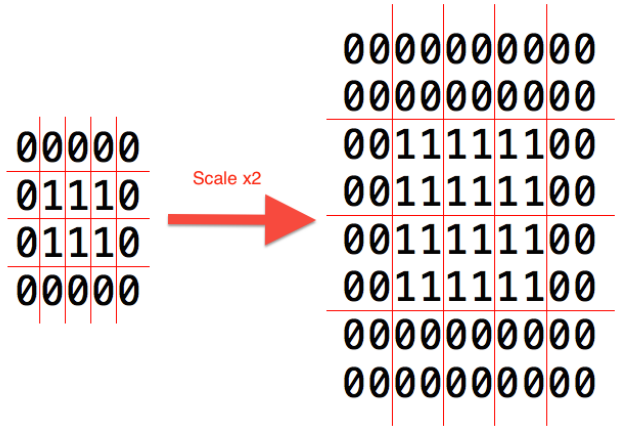



**Figure 4.4:** The result after scaling with a factor of 2

following relations for the index of the old values $(x, y)$ and new values $(x', y')$ where $s$ is the scale.

$$x' \in [x; x + s - 1] \quad y' \in [y; y + s - 1]$$

We then have to do every combination of $(x', y')$ in order to get all values in the scaled binary data.

### 4.3.3 Thickening

When we do thickening, we do not want the size of the binary array to grow any bigger. We just want the neighbours of any 1s in the binary data to become 1s.

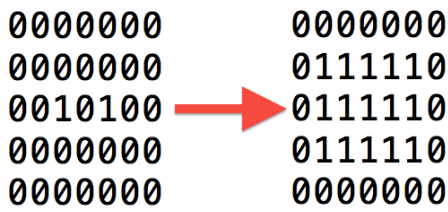



**Figure 4.5:** The result after one iteration of thickening

We start off by creating a copy of the reference binary array. We will only make changes is only in the copy binary, and at the end of each iteration we use the copy becomes the reference array. We traverse every field in the reference array and check if the value is 1. If it is, we will change its neighbours 1.

### 4.3.4 Thinning

As mentioned earlier, we want to go from a thick road to a thin road . To thin, we simply define the filters (Figure 3.8) as mentioned in section 3.5, and apply them. Additionally, we want the road to keep its width the way the snapshot is taken, so we don't omit this information. Instead of setting the thinned black pixel to 1, we increment it for each time the filter is detected. That means if we have the first case on Figure 3.8, the below number will be set to 2, indicating that the road width here is 2. We continue the thinning in both directions so we get the full picture (Figure 4.3b).

### 4.3.5 Zhang-Suen thinning

The way we decided to implement this algorithm, was to have an input and output array. Thereby any changes made to one array would not affect calculations. At the end of each iteration of the algorithm, all the values of the output array are copied to the input array, and we start over with the new values, until no further changes are detected.

Much like how the thinning algorithm is implemented, a difference here is that we take the neighbour pixels in an ordered sequence and check their transition from white to black. The conditions mentioned in section 3.5.4 will decide whether or not the pixel we are looking at will be white or black.

When running the zhang-suen algorithm, we use the iteration number to decide the width of curves we are thinning. So whenever we are changing a value, we give it the iteration number, as value. In this way 0 values in the array indicate a white pixel, where values over 0 indicate black pixel. An illustration of this is made on Figure 3.10.

### 4.3.6 Nodes and Edges

At the end of all binary manipulation operations, we create nodes and edges as instances of a class. We run through every position in the input array, and check for values that are greater than 0. We then create new nodes, and give them information like: id (place in the array), x and y positions, road width and scale. This information is stored for each node, and will be used later in the DFS implementation 4.4 in order to pass correct information to the roads. The road width is the value in the binary array.

In order not to have duplicates of nodes, we store them in a list, and we made an id check, to make sure a node would not be created twice. This way we can identify whether we should create a new node. If it already exists, we create an edge between it and its neighbours.

## 4.4 Depth First Search

During the DFS we go through every extracted node (no matter if it's a road or river), and send their stored coordinates to a list. However, in order to have smooth, and nice looking cohering roads, we need to handle a lot more.

The DFS is where all the complications occur, as it is here we handle the input data we get from the user, and need to analyse and understand the intention of this one.

### 4.4.1 Data structures

One of the hard parts of the implementation of the DFS, is keeping track of all the segment specific information. We identified the following:

- **All Points**: A list of list of points. One entry is a list of points, which constitutes the points of one segment, which are going to be sent to the curve fitting
- **All Road Widths**: A list of all road widths. One entry is a road width for a segment, which is found by taking the average of every road width contained in every node of that segment
- **Same Segments**: A list telling a segment if it's a continuation of the last segment
- **Intersecting Before**: A list telling a segment if it comes from a crossing
- **Intersecting After**: A list telling a segment if it is ending in a crossing

In order to have accurate information in the data structures, we made a method which updates all these in one time. This method was called in the following cases:

- **At the end of a DFS call**, which indicates the end of a segment
- **When there is more than the maximum amount of points in a segment**. The segment continues, but the method is called anyways
- **When meeting a crossing** (a node with more than 2 neighbours), which indicates the end of a segment

### 4.4.2 Crossings

Because of the thinning of the user input, identifying crossings was very simple. As explained in section 3.6 crossings are simply nodes that have more than two neighbours.

However complications arise, when making crossings fit with the roads they connect. In fact a crossing needs to keep track of which roads it is merging, and a road needs to know which crossing it is meeting.

### 4.4.3 Circle problem

One issue which was complicated to fix, was handling a circle as input, without the occurrence of crossings. As explained in section 3.6, we start in end vertices, meaning vertices with one neighbours. However in a circle there is no such thing. Therefore we were forced to start the search in a random node, and handle the fact that the search would end where it started.

The way we handled it, was by remembering the start node. If this node is met later in the same segment, it will mean we have a circle. We then add this node again to the same segment, so it appears as the start and the end node. And all we need to do after, is check what the first and last node is, when we create the road.

## 4.5 Mesh builder

Before we talk about how the implementation of the terrain is done, we need a way to create meshes in Unity in the most simple, elegant and intuitive way.

The MeshBuilder is an abstract class, which takes a set of vertices, UVs, normals and triangles and turns them into a mesh in Unity [Sur13].

These set of vertices, normals etc, must be given in a specific order in order for the mesh to understand their connection to each other. Triangles are also created by giving an order of vertex number, so the program can understand which vertices it needs to create a face from. The final class, *Mesh* [API14a], will then have all these vertices and indices for triangulation and create the corresponding mesh.

Unity uses 2-3 components to define a mesh in the environment:

- **Mesh Filter:** used to create the core mesh consisting of the vertices, faces and normals.
- **Mesh Renderer:** used to render the given Mesh Filter so the viewer can see the mesh.
- **Mesh Collider:** (optional) used to give a collider to the mesh, to create interaction in the game.

That means when we create the final mesh from the MeshBuilder, we create a `Mesh Filter` to create the mesh on a `GameObject`, show the mesh by adding the `Mesh Renderer` component and finally add a `Mesh Collider` to make it interact-able with the environment.

## 4.6 Terrain

The terrain is the most simple procedural mesh in our scene and its implementation is as simple. We define a resolution for the terrain, depending on how detailed we want the hills, smoothing etc., to be. And then, it is a matter of creating a triangle-strip with vertices, UVs and normals defined on Figure 3.13.

### 4.6.1 Height

We have two different data structures for separately taking care of the terrain height at a given position and the road height.

For our terrain, we use noise and turbulence functions (Listing 4.1) that gives us the nice and smooth terrain, and yet edgy on the surfacing, indicating cliffs and rock surfaces.

**Listing 4.1:** Noise and Turbulence functions creating the pseudo-random heights on terrain.

```
Noise(x,y)
{
  // ...Initialize freqs... //

  for(int i = 0; i < N; i++)
    h += Mathf.Sin(Vector2.Dot(new Vector2(x,y), freqs[i]));
  return (1.0f / N) * h;
}

Turbulence(x,y)
{
    for (int i = 0; i < 15; i++)
      sum += 1.0f/(Mathf.Pow(2.0f,i)) *
      Noise(Mathf.Pow(2.0f,i) * x,Mathf.Pow(2.0f,i)*y);
  return sum;
}
```

### 4.6.2 Storing heights

We use a special data structure for storing and getting the information of the following heights:

- **Terrain heights**
- **Road heights**
- **Environment distinction**

The first two are storing the heights of the terrain and road, respectively, meaning their float y-values. The environment distinction data structure is used as an overall image of what is positioned where in our environment. Values between -3 and -2, indicate a river, and values between -1 and 0 indicate a road.

These data structures have the same size as the amount of vertices in the terrain. So each vertex will have information about its height.

### 4.6.3 Road heights

The heights are mapped per vertex for the terrain and are applied to the vertices' y position. However the road has a different number of vertices and a resolution far different from the terrain. This brings us the issue that the road vertices needs to be mapped correctly to the terrain vertices, in order for the terrain heights to fit and smooth out underneath and the sides of the road. This is illustrated by Figure 4.6.

We first find the floor and ceiling values of the x position and then the z position we wish to check. Then by using all combinations of these, we get the four blue points shown on Figure 4.6.

The way we find the points to check is by taking the perpendicular vector $\hat{\vec{v}}$ to the direction vector $\vec{v}$. This ons is found by finding the vector from one mid point to the next (a midpoint is a point of the bézier curve). We then apply $\hat{\vec{v}}$ to the midpoint many times. The result of this can be seen on Figure 4.7. For each of these new points, we check the terrain for rivers and set its height appropriate to the road height. When going far enough out to the side, we smooth the terrain. That means we interpolate the height values between the road height and the terrain height.

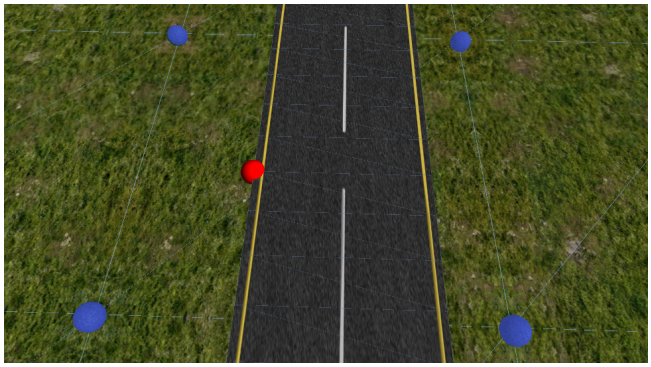


**Figure 4.6:** Vertex mapping. When the red vertex is checked, the vertex is mapped to 4 of the vertices on the terrain.

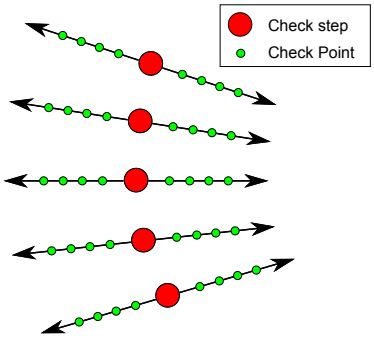



**Figure 4.7:** Checked points and the road bézier steps.

So taking Figure 4.7 as reference, the points in between the road width will all have the road heights. Whenever we move outside of the road width, we take the interpolated value between the road height and terrain height. The furthest point will then have the same value as the terrain height. By doing this, we ensure that the edges out from the road are smoothed.

## 4.7 Bézier

As the cubic bézier function is created by the step value $t \in [0, 1]$, we define a value which determines the precision of the curve. The number of steps is

iterated over and the corresponding x, y and z positions of the bézier curve is found, using equation 3.1. As the cubic bézier only affects the curves in the xz-plane, we do not set the y value. However this one is set by the random height function, further explained in section 4.6.3.

## 4.8 Pipeline

Almost all of our procedural objects are depending on the user defined path. In fact the road, tunnels and bridges all depend on the shape that is defined in the input.

The biggest challenge in creating all our path dependant objects, was creating the correct pipeline between these ones.

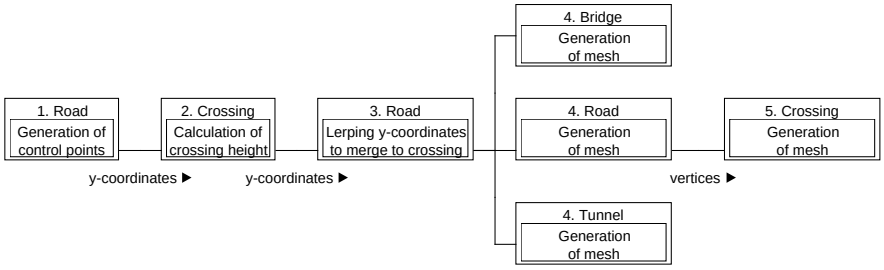



**Figure 4.8:** Pipeline of calling of mesh generation

As seen Figure 4.8 we have the following 5 steps:

1. We start off by generating all midpoints of the roads. After this, we have the initial (x,y,z) values of the midpoints. The reason we start by doing this, is because we have randomized heights for the roads, and transition to the crossings. The roads then send their first and last y-coordinates to the crossings
2. The crossings receive all these y-coordinates, and calculates the average of these, in order to find its own y-coordinate. Once this is found it sends it back to the roads
3. The roads then calculate a transition, so the heights match with the height of the crossing

4. With all midpoints having the right coordinates, we create the meshes. In this step we do all terrain checks to find out if we are to create bridges, tunnels or just simply roads. When all meshes have been generated, the start and end vertices of the roads are sent to the crossings, so it can build itself.

5. The crossing uses the vertices of the roads to generate itself, and thereby makes it look connected.

## 4.9 Road

As explained in section 3.9, we construct the road as a single triangle strip. However as seen just before we do the road generation in more passes.

### 4.9.1 Midpoints

The first step is receiving the midpoints from the curve fitting, and use them to create the shape of the road. This step is fairly simple, we advance little by little on the curve defined by the spline coordinates. As explained in section 3.6 the amount of steps we use for the road has been passed by the DFS class, and depend on the total amount of points used to calculate the midpoints. However those only give (x,z) coodinates, and we still need the y-coordinate. The y-coordinate is pseudo-randomly (as explained in section 4.6.3) generated so that it fits with the (x,z) coordinates around it.

In the end we store all the points in a list to use later. This list is added to a list of lists. This one is used to store all the segments so we can create the actual road meshes later.

### 4.9.2 Mesh

Now that we have all midpoints stored in the list of lists, we will focus on generating the road meshes of all segments, and making them fit together to seem as one.

#### 4.9.2.1 Road mesh

The way we create the road mesh (Figure 4.9), is by making the orienting vector from one midpoint to the next, however only in the (x,z) plane. This one is normalized, and multiplied by a road width. We then take its hat vectors (the perpendicular vectors), and apply them to the first midpoint. The two new points, are our vertex points and get stored in a list of all vertices.

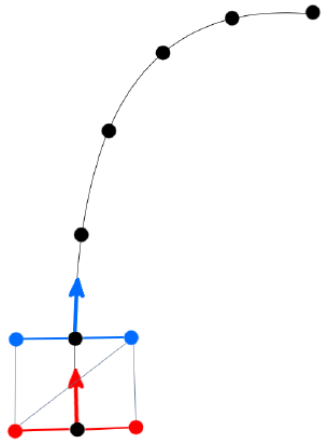


**Figure 4.9:** Illustration of the way we create a road mesh

The calculation of the texture coordinate is partitioned into two: the x and y coordinate. The x coordinate is either 0 or 1. It is 0 on the left vertex and 1 on the right vertex. The y coordinate uses the distance between two control points, divides it by a constant (to make it look fluid) and increments this value gradually.

#### 4.9.2.2 Transitions

In unlucky cases we can have a big difference between the last orienting vector of one segment, and the first orienting vector of the nextg. Therefore we remember the last orienting vector, and pass it to the next road segment. This one will lerp between the previous orienting vector, and the calculated one in the first three steps. The influence of the previous orienting vector gets smaller and smaller, so you don't notice the transition.

This method works as we take a varying amount of steps for the bézier calculation, so the distance between one point and the next is always close to being the same.

#### 4.9.2.3 Road widths

Having calculated the road widths for each segments, we are now interested in using the correct one. We decided to once again use a lerping function which would transition between the road width of the previous segment, this segment's road width, and the next segment's road width.

Given segments 1, 2 and 3, coming after each other respectively. The first half of segment 2, is used to transition between the road width of segment 1 and 2. The second half is used to transition between the road width of segment 2 and 3. Therefore you get the calculated road width of a segment only in the middle, and the rest is the transition between the different segments. Illustration of this can be seen on Figure 4.10.

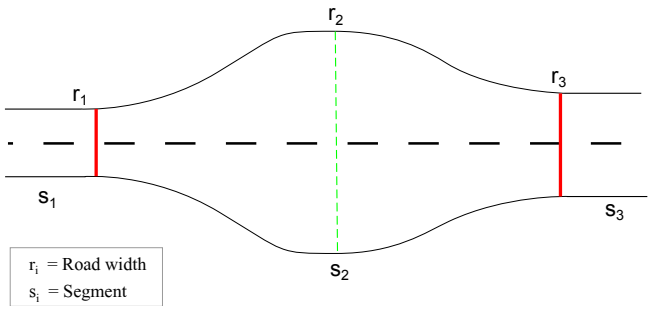



**Figure 4.10:** The three segments with their road widths interpolated.

#### 4.9.2.4 Road Types

In order to make sure the terrain doesn't cross with the roads we need to lower or raise the terrain to fit the height of the road (as explained in section 4.6). The goal is to control the terrain height and the road, in the areas where there is road, and change the type of the road, where the difference is too big.

The way we do it is by checking case of each check point. Then we check whether the point is higher or lower than the terrain. All those cases are stored,

and used to determine whether we should lower, raise or create meshes, like explained below:

- If all the points on the line is Case 1, we have a *bridge*
- If all the points on the line is Case 4, we have a *tunnel*
- If only one of the points on the line is Case 2, we *raise* the terrain heights
- If only one of the points on the line is Case 3, we *lower* the terrain heights

Once we have the main case, we know whether we need to create a new tunnel or bridge mesh. Otherwise we will lower raise the terrain. To identify whether we continue our segment with tunnel, bridge or road, we store the previous case number. If this case number, matches the case number of before, then we continue doing what we did before. For instance, if we were constructing a tunnel before, we will continue doing so.

## 4.10 Rivers and Lakes

### 4.10.1 Rivers

The implementation of the river is very similar to the implementation of the road. In fact we do exactly the same, except for creating a mesh. As explained in section 3.10, the rivers only appear as we lower the terrain under the zero-level, thereby revealing water.

For the river we do the same calculations as for the roads regarding the mid-points, the widths, the transitions and the lowering of the terrain. Only that the values in the lowering of terrain are set to different values than the road, in order for the terrain to be able to differentiate the two.

### 4.10.2 Lakes

#### 4.10.2.1 Identification

The way we identify lakes, is by checking (in the DFS class) how many points we send to the least square compared to the maximum amount of points we

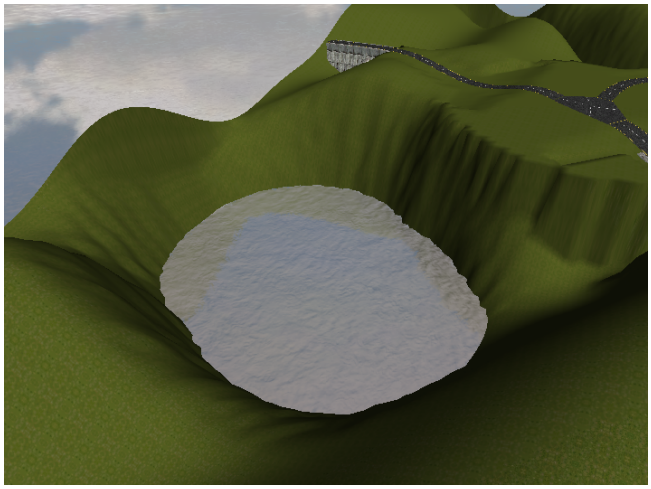


**Figure 4.11:** Image of lake in game

can send. If it is less than 1/5 then we do not call the curve fitting algorithm. Instead we take the point ($P$ on figure 4.12) that is in the middle of the list of points, and set all control points to this point. The river class will recognize that all control points are the same and that it should make a lake.

#### 4.10.2.2 Lake Bottom

The lake uses the river width ($w$ on figure 4.12) to identify how big the lake should be. This one is generated in the same way as it is for the road (section 4.9). The way the lake affects the terrain is similar to the way the river does, as it lowers the terrain to under the 0-level. However the approach is different. The goal is not to skip any terrain vertices.

As shown on figure 4.12, we make our lakes by taking circles of gradually bigger ray ($r$ on figure 4.12) to set the terrain height. We do not take every point on the circle to check with, however we take small enough steps, to make sure we do not skip any terrain vertices inside the circles. Therefore the ray, gets bigger by half the distance between two terrain vertices ($m$ on figure 4.12).

When traversing the circles to check points it is very important not to take too big steps. As $r$ is incrementing by $m/2$, it is sufficient of taking the steps on the circle of length 1. The circles are made by a combination of cosine and sine. Therefore we increment the angle, and need to do that with care, as the

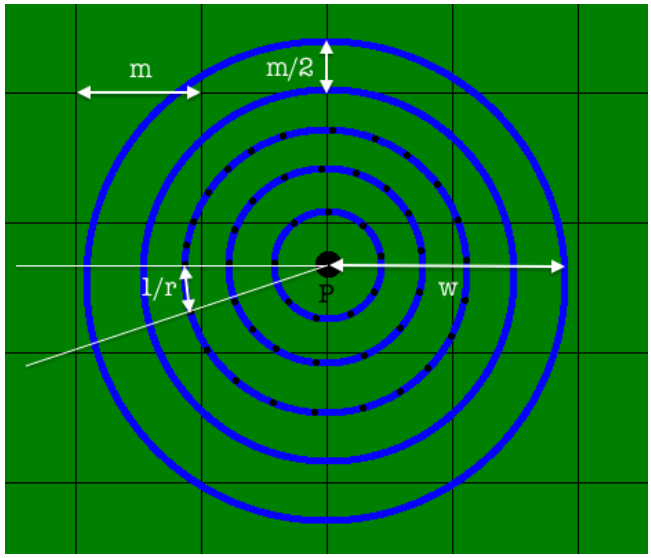



**Figure 4.12:** Image of the way we make lakes

bigger the ray, the smaller the angle should be, in order to take steps of the same length. By setting up the relation that we wish to take steps of length 1, it means we will need to take as many steps as the size of the perimeter of the circle in question. Hence we take $2\pi r$ steps. Now looking at angles, we have the angle $\phi \in [0; 2\pi]$. However we wish to partition that in $2\pi r$ steps. Therefore one angle step will be of size $\frac{2\pi}{2\pi r} = \frac{1}{r}$.

Figure 4.13 shows all the points we check for a lake. The white points are the ones to se the bottom of the lake, the red points are for smoothing the terrain.

#### 4.10.2.3 Transition

For smoothing the transition down to the lakes, we do the same but with double the width. Just like with the smoothing of river and road, we reduce the smooth factor the further away we get from the actual lake. Assuming $d$ is the distance from the lake to the point we are checking, the smoothing factor will be $d/w$, where $w$ is the width of the river.

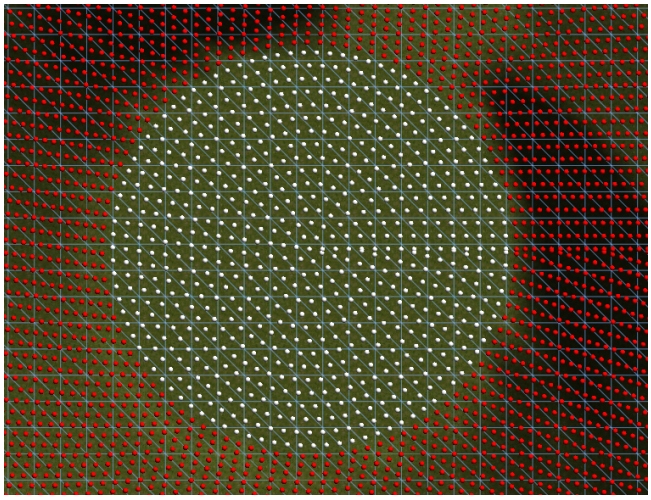


**Figure 4.13:** All the points we check when making lakes

## 4.11 Tunnels

For each of the midpoints from which the tunnel is created, we create a new set of vertices in the upward direction. This direction is run through as a half-circle by use of the trigonometric functions. More specifically, we take the cosine and sine of the iteration number divided by the density until we have a half circle:

$$\vec{v} = \left(\cos\left(\pi\frac{i}{d}\right), h\sin\left(\pi\frac{i}{d}\right), \cos\left(\pi\frac{i}{d}\right)\right), \quad i \in [0; d]$$

where $h$ is the tunnel height and d is the density of the tunnel.

In the `MeshBuilder`, we connect the vertices by the density of the curvature of the tunnel, which we define. The higher the density, the more vertices and faces.

### 4.11.1 Entrance

When we identify the start or the end of a tunnel (see Section 4.9.2.4), we know we have to create an entrance. The tunnel sends all its midpoints to the entrance class. The shape of the entrance is like a box wrapping around the tunnel (Figure 4.16). The vertices are created by making a rectangle, and these

ones are mapped to the tunnel vertices. The side vertices then connect to their neighbours in order to create faces. Likewise with the top vertices (example can be seen on Figure 3.20).

### 4.11.2 Hole

For the hole, as mentioned in the design section, we set the terrain vertices at the tunnel positions to the same as road height, but we store these positions in a list. The list contains the mapped vertices, we got when checking for road height. This means that we also will receive duplicates of vertices, because the density of the road is different (much greater in our case) than the terrain density.

The way we deal with all these duplicates is using the C# Library `System.Linq`. It is able to remove duplicates in a list. Additionally, it has a sorting function, which is useful when we want to run a triangulation over vertices. So when we want to use triangle-strip, we sort them by their z-values first, then by their x-values.

This list is used to decide on the creation of a tunnel cap. The tricky part here is to connect them to the ground where the road is beside the tunnel entrance, so the hole of the tunnel is created. This is done by taking a distance out from the midpoint, in between the entrance wall and the tunnel wall (Figure 4.14).

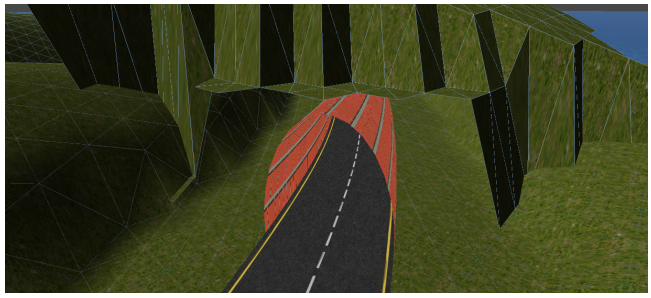


**Figure 4.14:** Tunnel entrance hole. The vertices opens the hole, whereas the entrance gap is hidden by the entrance (Figure 3.20)

The tunnel cap is then covered by the entrance. The mesh vertices of the tunnel cap are connected separately, by taking each vertex with their defined positions. As we have them sorted by their z-values and then by their x-values, we can run upward from a triangle-strip and connect them. See illustration on Figure 4.15

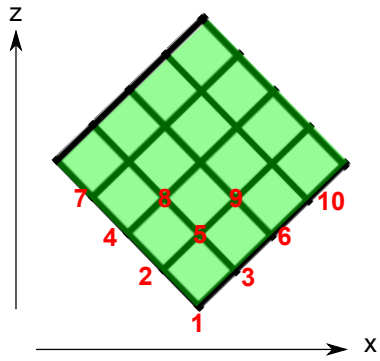



**Figure 4.15:** Creation of tunnel cap. As they are sorted, the vertices are created by their orders given in the picture.

The final look of the tunnel cap can be seen on Figure 4.16.

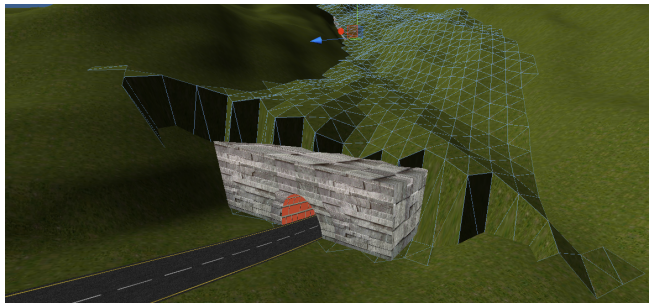


**Figure 4.16:** Tunnel cap.

## 4.12 Bridges

We can partition the implementation of bridges in two parts: identifying bridges and creating the actual bridges.

### 4.12.1 Identification

Whenever we set the values in the *Environments distinction* array (section 4.6) when working with roads, we do some checks first. We check whether the value inside the array is smaller than the one we wish to insert, but more importantly we check whether there is a river or not. If the *Environments distinction* array indicates that there is a river, the road will not set the height, but make a bridge instead.

### 4.12.2 Creation

In order to create itself, the bridge needs dynamic information from the road, as every bridge is specific to the road it will raise. We will separate the creation in several parts: the top and the bottom.

#### 4.12.2.1 The top

The top is very trivial, as it is simply a wider road. So the road sends the control point, the orienting vector and the road width. The bridge then multiplies the roadwidth to find its own and creates its top like a road.

#### 4.12.2.2 The bottom

Having defined its top, we need to make the vertices under them to make a side and bottom. We therefore made a curve that would define the height of the bottom vertices, as shown on the black line of figure 4.17.

The line is used in a symetric way. This means we traverse the curve from left to right first, then from right to left, then from left to right, and so on. This is done both top vertices.

With all vertices in place, now we make the faces as shown on figure 4.17, all vertices of the same side (top, side, bottom) are connected together.

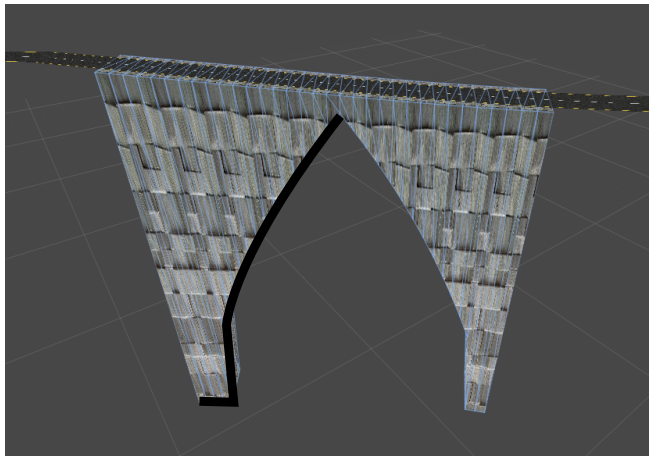


**Figure 4.17:** The bottom height curve

## 4.13 Crossings

The crossings meshes are not complicated to construct. However in order to make them fit and look good together with the road it gets a lot harder.

### 4.13.1 Sorting Vertices

As explained in 3.13 we are using the road vertices meeting the crossing, as vertices for the crossing. Therefore these last ones are coming as pairs.

The first thing we do is sort the vertices according to an orienting vector. This orienting vector is found by making the orienting vector from the second last point to the last points found in the DFS (opposite from the first to the second when starting in the crossing). We divide the x and y values by themselves so as to get one of 8 possible orienting vectors (figure 4.18).

Having the vectors, we sort the vertices according to which orienting vector the road had. This way we assure the right vertex of one road will connect to the left vertex of the road on the right, and so on.

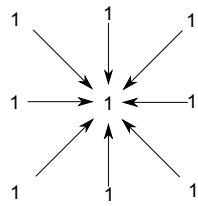



**Figure 4.18:** The possible crossing orienting vectors

Chapter 5

# Test

In this chapter, we are going to present our way of testing our implementation of the procedurally generated environment. We are going to look into some test cases first, where we later mention how we handled errors when testing and also, what our strategy for solving them was. This chapter will also include the interviews we have done during our analysis of our design. Finally, we will include user tests.

## 5.1 Input

Our main test cases for our program come from the camera input, which is dependant on the resolution of the camera. But we decide to scale the images down to 160x120 pixels, no matter the resolution (see 4.3 for explanation).

For practical reasons, every time we needed to test a particular road drawing, we had to use our device camera to capture the same image over and over again. This became redundant in the end, which lead us to make a read and write methods of pixels we get from the camera. We store these pixel data into a file, which can be reused all the time. We use `System.IO` from C# library to achieve this.

The great part of this is that we are able to test exactly the same input several times, in order to find out whether we have fixed the problem or not.

We have many different test cases in order to justify the requirements of our project. Starting simple, we have a straight line. We test whether our program is able to read the straight line, so first step was to see if we could convert them to pixel data set of 160x120 pixels. We output the files as shown on Figure 5.1. An example of a capture and a comparison of that to the virtual environment

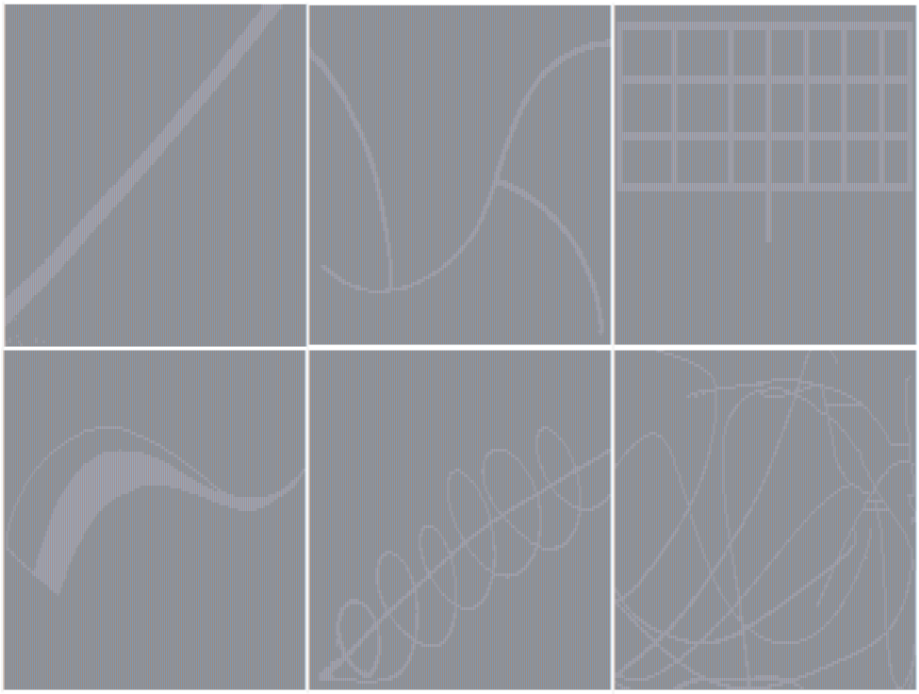


**Figure 5.1:** Test cases.

is shown on Figure 5.2. We view the environment in x-z plane to match the picture.

### 5.1.1 Error handling

First of all, we check whether our drawing matches with what is read by looking at these files. Of course, we do not always get as clean images as in Figure 5.1.

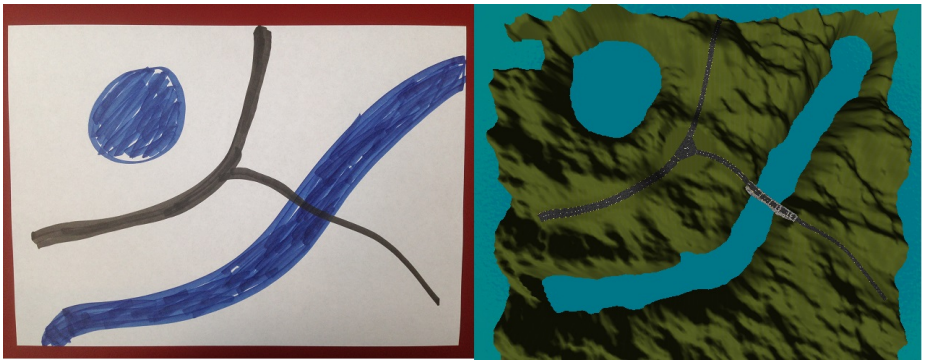


**Figure 5.2:** Compare the "Environment" test.

The reason is because of the lighting in the room, camera focus or other factors distorting the image.

When we approve our image with the output, we check whether our thinning algorithms have worked as we expected them to (as we check the input by their distribution of numbers, see Section 3.6). Finally, when the binary extraction works along with the thinning algorithms, we start generating our environment from this dataset.

We have to get the expected visualization from the given input. Things that can go wrong here are

- Deformed mesh faces (wrong indices or combination of triangulation)
- Missing vertices
- Orientation of the faces (including back-faces)
- Overlapping faces
- Wrong vertex normal vectors
- Incorrectly mapped texture coordinates

On top of these visually incorrect output, there are the programming errors such as `NullpointException`, `IndexOutOfRange`, `MissingComponent` etc. The way we solve these issues is by iteratively fix the errors as they appear until the result looks satisfactory. An example of one error we had to fix was the road overlapping the terrain vertices at certain points as shown on Figure 3.17.

Here, we needed to find out what was causing only some of the vertices to be above the road, while it worked well on other places. By careful debugging, trying out different cases, we tried to pinpoint the problem. The reason turned out to be a rounding problem when mapping road vertices to terrain vertices. The issue was fixed by checking both flooring and ceiling of vertices and then set their corresponding road heights.

## 5.2 Interviews

The interviews were conducted by meeting up with the users personally. In the interview session, as we were two persons interviewing, one of us was responsible for asking and interacting with the interviewer, while the other one was taking notes and supplementing with questions. The interview questions with their answers and summaries can be seen on Appendix A.

The results were very diverse, but most of them agreed upon some common subjects of level design. We have collected and extracted the most important information from their summaries, which we marked with a yellow marker in the appendix. We tried to give each of their ideas of features for our product into a title and mark them. Then we counted how many of the interviewed people said the same and the most occurrences of ideas can be seen on Table 5.2.

| Ideas | Marta | Johan | Martin | Peter | Ole | Mikkel | **Total** |
|---|---|---|---|---|---|---|---|
| Play the level | x | x | x | x | x | x | 6 |
| Obstacles | x | x | x | x | x | x | 6 |
| Paths and Patterns | x | x | x | x | x | x | 6 |
| Start and End Point | x | x | x | | x | x | 5 |
| Assets | x | x | | x | x | x | 5 |
| Make puzzles | x | | | x | x | x | 4 |
| Real-time | | x | | x | x | x | 4 |
| Model Landscape | | | x | x | x | x | 4 |
| Tweakable Character | | | x | x | x | x | 4 |
| Heightmaps | x | | | x | x | x | 4 |

## 5.3 User Reviews

Apart from the preproduction interviews, we have also postproduction user tests. This gave us an indication of how well the program is working, and if

there is an interest in the concept.

We asked Konrad Stanek, with whom we had made our previous bachelor product for [Jø12], to test the concept. We asked him to give us his thoughts and opinions about the product.

> *"We had a pleasure to collaborate with Patrick and Bilal in 2012-2013, when they developed the first version of pseudo-random scenes generator for simulated car driving. We are currently using the platform for a range of cognitive experiments, where participants are asked to take various voluntary decisions related to driving a car, while the EEG and fMRI signals are acquired from their brains.*
>
> *The new version of the scene generator, presented in this thesis, seems to be an ambitious and interesting extension of the original platform, offering much faster and easier interfacing with range of graphical or numerical software packages, and even hand-written drafts. Furthermore, we observe improved graphics and realism of the generated scenes. As such, the platform may have a wide potential application primarily in gaming industry, but also in research areas involving cognitive/behavioral tasks performed in virtual environments (where number of pseudo-random scenes needs to be generated automatically).*
>
> *Of course, for the latter application, the generator needs to accept certain input constraints, such as road width/length, visibility, car/avatar dynamics, etc., just as it was achieved at the initial version."*
>
> - Konrad Stanek

We also asked Marta La Mendola, one of the level designers we had interviewed, in the early stages of the project.

> *"I think it's very simple and fun to use. It creates something really fast to test and play around with. It could be used for a driving game where you can create your own maps and challenge friends. It would be nice to have obstacles and really simple enemies. Maybe depending on the colour it can be an enemy."*
>
> - Marta La Mendola

We also presented our work to a few high school students. We told them about the concept, and showed them how it worked. They were amazed by the idea, and the fact that it could generate environments from just paper. They said it was great idea, as everyone can draw, and therefore easily make levels. They also asked us if we were going publish it on the android or app store.

Chapter 6

# Analysis

In this chapter, we are going to discuss the results from the implementation and test chapters. We are also going to discuss what parts were successful and unsuccessful.

## 6.1 Outcome

We are satisfied with the idea we have come up with. It is a new and innovative idea, and we believe it could entertain a lot of people if developed correctly. Our contribution however remains a prototype, so a lot of work is needed for it to have a commercial value.

The tunnels and the caps above them are not working as well we would like. We have not found a solution to integrating tunnels perfectly, and having them go through terrain in a realistic looking situation.

The structure of the code would be something we would change. If we were to start over, we would probably change a few things, in class dependencies, and data structures. For example we have a lot of segment specific information we are storing in different lists. Instead we would make a struct with all this

information, in order to make it a lot more extendable and easy to handle.

## 6.2 Creative process

We believe the creative process was executed correctly. Brainstorming is a powerful tool to get ideas, the interviews gave us great feedback, and the meetings with our supervisor kept us on track and made us focus on the right tasks.

If we were to redo the project, we would probably separate the project period to idea development then implementation more than we did. We would probably start earlier with the interviews, as the outcome enhanced the quality of the concept a lot, and really boosted production and inspiration. It taught us the importance of interviewing, and interacting with experts.

## 6.3 Work process

In a small team of two people, planning and separating work is not as important as in bigger teams, however we found it very useful, and is definitely something which is not negligible. As explained before, most of our effort in the beginning of the project period, was put on the concept development. And as we moved along, and the concept gradually got more concrete, more and more time would be spent on implementing.

We had an agreement to meet as often as possible to work together. The fact that one person is working is motivating the other to work as well, and in this way it had a motivating factor on whoever would be falling in productivity. In the beginning of the project period, we would work together in pair programming. This allowed both of us to get acquainted with all the code and structure as it was created, and for both parties to contribute and affect the outcome. As we moved along, as tasks would become more specific and separable, we would separate the tasks, and thereby double the production speed. Of course some time would have to be spent filling the other person in on progress and structure changes, which we did very well. Thereby making us experts in different areas of the program, without losing the total overiew. Also it would mean both parties could be productive at different times and locations, and working from home became a possibility.

One flaw we had was the accept that things take time, and has risks of not finishing in deadlines. Instead we should have been better at giving each other

deadlines, as we work well under them. We also learned that drawing things, and writing initial algorithms on paper is a very powerful tool. It is a solid way to explain ideas, and to fully understand them yourself, so this method would have been used a lot more had we known it from the start.

### 6.3.1 Unwanted results

Unfortunately, the result from the tunnel cap explained in section 4.11.2 did not make the terrain smoother, neither connected in an admissible way. First of all, the vertices did not smooth out from the cap to the rest of the surroundings. Secondly, if we had a very small tunnel with very few segments, the hill made little sense as we had a mini triangle-strip and would be better off without. Thirdly, we did not know how to connect the side vertices with the rest of the terrain in a sensible way.

If we had more time to re-analyse and design our implementation, we would have found a way to make two meshes intersect and make a hole, where the tunnel vertices are. For the smoothing, we could have made the mesh connectivity dependant on other vertices, so their heights are created in the same way as the height of the terrain.

### 6.3.2 Evaluation of user reviews

We have had a few users review our product, as mentioned in the test section 5.3.

Konrad reviews our product as having potential, primarily in the game industry. This is positive, as our aim was to make a game. Additionally, he does not exclude the possiblity of it being useful elsewhere, like in research facilities needing automatic generation of virtual environment. This opens another door of possibilities for our product to become commercially valuable.

Marta sees the intuitiveness and playfulness in our product. However, she restricts the program to be used for racing games. She sees possibilities in improving the generation by adding obstacles and enemies, which we believe would make the levels created more fun, and bring out the playfulness of a user creating his own levels.

Lastly, testing on high school students, gives us the possibility to see if the concept is of any interest. They have an outside point of view, as they have

not been included in the development process. The fact that they ask us if we are going to sell it, as well as suggesting us to publish it, shows that they see potential in the concept.

To sum up, all these user reviews give us positive signs that our product can be used in digital research and entertainment.

## 6.4 Extensions

If we are to continue working on this project, these are the things we would like work on:

### 6.4.1 Binary Extraction

Image analysis is not our field of strength, nor was it our focus in this project. However there is a lot of work to be done, and a huge quality improvement if we were to work on this some more.

**Calibrating**: We could do some calibrating of white and black, so as to be better at recognizing them, and reducing a lot of errors and misgeneration from the lack of light.

**Easily recognizable shapes**: is also something that could be interesting for the user to have as option. Things like squares, circles and crosses would give the user a new dimension to the levels created. Those could be obstacles as well as events, mops or pick-ups. Alongside this we would make an infastructure that would allow the user to decide what each shape represents.

**Vegetation areas**: We would have liked to introduce green as a color code on the paper, indicating vegetation areas.

### 6.4.2 Optimizing

**Algorithms**: We have always worked with care, when making our algorithms, therefore all of them are performant. However we are convinced a lot of smart rules can be made in order to save calculation, and we believe it could fasten the generation, especially when it comes to thinning and scaling.

**Intelligent roads**: We could have the program try to understand the curves drawn, and draw conclusions from it. Roads that are close to each other with many sharp turns, could mean they are climbing mountains, whereas straight roads would mean highway.

### 6.4.3 Different Meshes

**Bridges**: as mentioned in section 3.12 we would have liked to make different types of bridges for the different purposes.

**Roads**: with very little effort, we could have many different kinds of roads. We could add side walks, lights, fences and other objects.

**Vegetation**: Several kinds of vegetation could be implemented and added to the program. Namely grass, trees, bushes...

### 6.4.4 Styles

Something which would make our program a lot more fun to work and play around with, would be the possibility to change the styles, thereby enhancing diversity.

**City**: With this style one could imagine normal highway roads, with obstacles being buildings, and vegetation being normal trees.

**Western**: In this style we could have path like roads, worn out buildings as obstacles, as well as fences, cacti, dead trees, and a desert terrain.

### 6.4.5 Game modes

In accordance to the different styles of meshes, we could have different game modes.

**Racing**: This is our main game mode, where the player can drive around in his terrain. We could add checkpoints and pickups around the tracks to pick-up, for the player to win.

**Third person**: A game mode where the player has a lot more mobility and freedom to move around on the terrain in the way he pleases. In this mode there could be mops around on the terrain for the player to find and eliminate.

**Multiplayer**: Allowing a user to play with friends on environments they have generated would be of big interest. One could make an arena like game, where the players fight in the environment created, or a racing game, etc...

### 6.4.6 Graphics

Unity provides a lot of graphical improvements, that can be applied after the environment has been generated. This means that the performance of the generation would not be affected.

**Shaders**: We are already using shaders for water. However we can improve the shaders of the car, and the procedurally generated objects. This would make the scene look better, and be more performant.

**Shadows**: The scene will look at lot more realistic, if we cast hard and soft shadows on the terrain.

**Lighting**: We could focus on making lights on the scene. It could be car lights or light from lampposts beside the road or inside the tunnel.

**Fog**: We can introduce fog and other environment details such as halo from the sun or particles in the air.

## 6.5 Credits

The content of this report and the product developed during this project has been done in collaboration of the authors. While some subjects and work were divided and the focus lied in different aspects of this project, either of the authors have touched upon all the parts.

### 6.5.1 Work by others

This project includes work from others as well as our previous work in our Bachelor Project [Jø12]. The fact that we built upon our Bachelor Project made us reach further in our development and focus on other important parts of this project. Here are the things we cannot take credit for:

- **Car**. The car model, movement, camera etc, belonging to the car driving experience, is from Unity Tutorials. It was taken to make it easier for us to create a playable environment.
- `MeshBuilder`. Developed by Jayelinda [Sur13]. The `MeshBuilder` is an easy tool to make indices and triangles for building the meshes.
- **Water shader**. To resemble water in Unity, we had used Unity Pro's water shader and prefab. Alternatively, we could have made our own water shader.

Chapter 7

# Conclusion

We have created and developed a new approach to level design. We give a solution that allows non-technical people to easily and rapidly make their own levels and feel ownership over these creations: simply by drawing lines and curves on a piece of paper.

To directly connect and conclude the main problem statements:

- We can generate an environment from a 2D sketch with a lot of input variations. We take into consideration input reading errors, and generate different forms of pseudo-random heights to terrain, roads and other objects.
- We can generate the environment in real-time and are able to implicitly customize different parameters (such as road widths) to give more control to the user.
- The program can generate levels for a racing game, and with some changes it could be adapted to other game types. We have also tested our concept on users, who see it as a potential entertaining product.

We have experienced the process of going from a simple idea, to getting a well-defined product. In the process we had to redefine the vision, interview people from the target group, and change the idea to match the new input.

Although the program itself can be worked on a lot more, we have still managed to make a solid prototype which illustrates the concept, and allows users to draw shapes, and recognize them in the roads and rivers created. We also made sure the interaction of these made sense with the introduction of bridges.

With further development of this program, and a nice set of rules, we believe it could be a fun mobile app, that could be enjoyed by many, thanks to its intuitive approach and to its coherent output.

Appendix A

# Interview

We interviewed 7 different people which fit our target group as level designers (one of them was a Game Designer). The names and their occupation of the people we interviewed are

- Marta La Mendola, Student/Intern IO Interactive
- Johan Buhl, Student (Game Designer)
- Peter Buchhardt, Level Designer at Playdead
- Astrid B.Z. Madsen, Level Designer at MovieStarPlanet
- Mikkel Martin Pedersen, Level Designer at PressPlay
- Martin Vestergaard, Student

## A.1 Interview questions

Our project's target group is level designers. When level designers work, they usually make rough sketches of levels. Our idea is to allow level designers to visualize levels through our program.

---

Template

- *Question*
  - What we want from the question

---

- *Did you find the idea of the project interesting?*
  - Getting initial ideas from level designer
- *What are your impressions after watching the video?*
  - What does the level designer see as initial opportunity for the tool
- *Do you use any tools for level design?*
  - What features of a level design tool is interesting for the level designer
- *How do you draw your initial level on paper?*
  - Learning what to identify in our program
- *What's a normal creative process of level design like*
  - Where in the level design process is it interesting to have a tool like the one we want to make
  - Understand how level designers find their ideas (how they work)
- *How do you tweak a level?*
  - Understanding the requirements for us to being able to make a tool which allows level designers to tweak levels.
- *Once the creative process is done, what are the next steps for you?*
  - On paper, imagination, other...
- *What do you see as the bottleneck in level design?*
  - Is there a feature our program can do to hasten this process.
- *Do you ever have a problem making others understand the vision you have for a level?*
  - Can this tool be helpful to create a unified vision in a group?
- *What is a level editor for you?*

  - To see if we fulfill the requirements and if we missed an important point

- *What do you expect from a level editor?*
  - To see if we have all the features in place needed to make it useful
  - To enhance the quality of the product focusing on the most essential parts

### A.1.1 After using the program

- *How did the program meet your expectations?*
- *What features would you have liked there?*
- *Can you see this being useful for level design?*
- *Could you imagine yourself making levels using a similar tool?*
- *Would it be an advantage for you to being able to put objects your own objects in this visualization tool?*

# Marta - Summary

- *Did you find the idea of the project interesting?*
- *What are your impressions after watching the video?*

She liked the video and the idea of the project. She liked the aspect of being able to test ideas that you get on the fly, by generating levels fast. She thinks it's nice to have a tool that lets you play around with a level, and make the team understand what your vision early in the process.

- *Do you use any tools for level design?*

She mostly used paper and pen. For big projects, she uses software like UDK. It is similar to using the Unity engine but more oriented for level designers (not technical people). It is a very complex tool, which takes time to get into. Only placeholders for the different interactive elements in the levels is built. It allows visual programming for things like AI. Photoshop can be an alternative to paper and pen.

- How do you draw your initial level on paper?

She said it depended on what type of game you are making. Then as an example, she mentioned that if it was a platformer, you could start marking start and end points and sketch a path between them. Then you can place different objects and make a puzzle out of them.
She also mentioned that it depended on what type of level designers you were. She mentioned two kinds of level designers: Art oriented (herself) and programming oriented. Some level designers do not know how to draw. It is important for a drawing, to make everything understandable, not only to yourself, but also to others. She mentioned that it would be nice to make puzzles, and have different events, and obstacles that can be placed.

- *What's a normal creative process of level design like*

She mentioned that the creative process of creating a level depends a lot on the type of games you are making. She said that there were wide possibilities.

- ❑ **It's important to know your limits:** The game designer defines the mechanics, thereby defining the limits of a level.
- ❑ **Sketching ideas:** Use many tools to generate levels fast and test them.
- ❑ **See what works:** Take the things that work, and extend those ideas.

But the creative process is all the way through a game development process, and not only in the beginning. It is only the core mechanics, and the fundamental puzzles in the beginning.
A lot happens on paper. Not only drawing of levels, but also strategies and much more.

- *Once the creative process is done, what are the next steps for you?*

She said it takes a lot of time to build a level and something to test on, because it takes a lot of time to setup. She meant that it did not matter whether the level look good or bad. She mentioned that on UDK, it takes time to build levels which takes time from testing the player experience of a level.

- *What do you see as the bottleneck in level design?*

She had different experiences depending on the times and conditions she had to build levels. In DADIU they all worked at the same time. In studios, they plan the process so the level designer comes later in the process. She groups everything together that comes from the programmers and artists. She said that level designers connected everyone in the project and nobody had to wait for her. She works on a level until she is satisfied with it. At the beginning of her time in DADIU, she had a hard time getting started, because the concept was unclear. It would have been nice to have something that would unify the game and level designer. After a session that had the aim to establish the story of the game, they got a unified vision of the game. It would be nice to have a tool that could do this for mechanics or gameplay. She mentioned that randomly generated buildings or squares, start and end points, a timer, being able to test, and obstacles would all be nice features to have.

- *Do you ever have a problem making others understand the vision you have for a level?*

She thinks it's hard to make others understand your ideas, as people think very differently. It is much more useful to draw ideas on paper, as it gives a concrete reference. She mentioned that a visualization tool allows everyone to understand better, as there is less to interpret.

- *What is a level editor for you?*

A visual tool that helps creating levels by defining the area the player interacts with the environment or the elements of the game. Additionally, it would include different assets, static props and dynamic elements. She mentioned that it could also be used as a visual tool as it helps you create the environment, both from inside and outside.

- *What do you expect from a level editor?*

She mentioned different level editors to relate with, such as Hammer (Valve) and UDK. She said it should be able to build an environment with paths, ladders, and a lot of other stuff. She also mentioned that it would be nice to have visual programming with easy scripting included.

Hammer (Level editor) Left for dead 2 levels.
Building the environment.
Building the paths. Ladders. Visual programming, script easily. Create a lot of stuff.

***After using the program***

She said it would be nice to being able to test on both the smartphone and on the computer. Therefore exporting to unity would be nice. She also wants to be able to draw a line straight from the phone instead of on paper first. If you want to draw something quick, go straight on the phone. She also wants to be able to put numbers and signs around, mapping events on the program to get the idea of a level. And this should be done both offline, and during runtime of the program.

- *How did the program meet your expectations?*

Close to expectations and excited. Interested in testing.

- *What features would you have liked there?*

- ❑ Play the level,
- ❑ Adding events (mapping markers and stuff)
- ❑ Different colors of roads (Color code)
- ❑ Different types of markers
- ❑ Timer
- ❑ Obstacles
- ❑ River
- ❑ Tunnels

- *Can you see this being useful for level design?*
- *Could you imagine yourself making levels using a similar tool?*
- *Would it be an advantage for you to being able to put objects your own objects in this visualization tool?*

She would like to being able to make simple shapes (cubes, spheres, etc.) and perhaps bridges, and the possibility to make own objects.

# Johan Summary

- *Synes du at ideen bag projektet lyder interessant?*

Han forstod ikke helt hvad konceptet gik ud på. Spørgsmålet er hvad for en type spil det er man vil lave.

- *Hvad synes du efter at have set videoen?*

Toolet skal laves til en specifik genre, den kan ikke fungerer på alle spil.
I konteksten af en level editor for racer/platform. Start med en genre, og så udvid til flere.
Han vil gerne have muligheden for at definere standard ting: loop, start, slut, forhindringer.
Symboler der kan identificeres af programmet til at indsætte objekter.

- *Har du nogensinde brugt nogen tools til level design?*

Unity.
Hjemmelavet tool til hans egen spil. Bruger heightmaps til at placere voxels i forskellig højde.
Han forklarer at AR er et game element frem for noget man ville bruge som tool. Der er noget eksplorativt ved at bruge AR. Han er meget interesseret i at alt sker i real-time, så assets dukker op.
Han er ikke glad for AR generelt. Det er irriterende at holde kameraet i fokus. Det skal virke hver gang.

- *Hvordan tegner du et udkast til et level på papir?*

---

- *Hvordan foregår en normal kreativ process af level design*

Processen er anderledes alt efter hvilket spil man laver.
Punish Panda som eksempel. Han går efter en spiller følelse, idet spillet lægger op til gåder. Han vil gerne fremprovokere Aha følelsen.
Han tænker på hvad det er for nogle ting man har at gøre godt med, og laver en skitse ud fra det.
Idé: det er sjovt at pandaer falder ned på en rundsav. *Han tegner det, og viser at man starter med en simpel idé, og laver udfordringer efter*.
Bilspil som eksempel: Han tager en almindelig bane, som han synes er lidt kedelig og tilføjer nogle smutveje. *Han tegner en bane, med store sving, slalom sving, grene, skarpe sving. Så kommer udfordringer, jump pads*.
Han nævner at det er vigtigt at definere hvad spillet er, hvad for nogle mechanics man har at gøre godt med. I et tidligt stadie er det ok med placeholders, det er ikke vigtigt at have specifikke objekter, så længe man ved hvad de enkelte er. På papir bruger man tegnsystemet til ens egen

forståelse, og dem man arbejder tæt sammen med. Men det også vigtigt at få alle til at forstå levels'ne, men det er først længere inde i projektet.

- *Hvad er skridtet efter den kreative process?*

--

- *Hvad synes du er den del af level design som trækker level design processen ud?*

At bygge levels er dét som tager tid, men han gider ikke (han er game designer). Men det sjove er at teste. Dog ved man ikke om noget fungerer før man har testet det på en endelig version (Fx flere levels i Punish Panda blev slettet fordi de ikke var sjove).

- *Har du nogensinde problemer med at få andre til at forstår den vision du har af et level?*

- *Hvad er en level editor for dig?*

Den skal kunne lave baner, have adgang til et bilbiotek af assets som er nødvændige til et level.
Der skal nok frihed til at det ikke begrænser designerens kreativitet.
Man skal finde den rette grænse mellem at have adgang til det hele, og adgang til lidt.

- *Hvad forventer du af en level editor?*

**After using the program**

- *Hvad synes om programmet?*

Han synes at AR-delen var lidt besværligt at arbejde med og at der skal være en grund til at bruge det. Men han udelukkede ikke at den slet ikke kunne bruges til nogen andre typer af spil. Han synes at det var en smart lille gimmick, men ikke et værktøj. Han synes virkeligt godt om at man kunne tegne en road ud fra et tegning.

- *Hvad for nogle features kunne du godt have tænkt dig der var?*

Han nævner, at VR (nævner Oculus Rift) kunne være en mulighed for at have som en god feature til test og se banen fremfor AR. Han mener at VR er vejen frem.

- *Kan du se brugbarheden af programmet til level design?*

Han mente at generering af veje fra et stykke papir er et godt tool til hurtig at se og vise levelet til andre.

- *Kunne du forestille dig at lave levels ved brug af et lignende tool?*

Han nævnte generelt om at generering af level ud fra en tegning. Han så tablet-delen mere som et teste-redskab fremfor en 'level creator'.

- *Ville de være en fordel at kunne lægge dine egne objekter i dette visuelle tool?*

Ja og han mente at hvis man kunne køre det i realtid samtidig med at man havde nogle brikker at flytte med, ville det give en mulighed for at designe levels.

# Martin Summary

For finding inspiration he uses Google maps to find interesting looking road paths and patterns. He uses this to draw initial paths, and works from there. Afterwards he builds on it, tweaks it, finds mysterious routes and many different patterns. He uses the drawings for himself, and contain many symbols and patterns, and only he can understand.
He uses a game specific level editor, and uses many different patterns and structures in it.

- *Did you find the idea of the project interesting?*

He finds it an interesting project with potential. As a level designer it is hard to show your ideas to others. He likes to draw, and would find it very interesting to have a visualizing tool, many good ideas can be visualized.
Many things are complicated, to do separately.
He sees many possibilities in the project.

- *What are your impressions after watching the video?*

After reading the mail he was a very confused. After having watched the video and reread the mail, it made more sense.
It is very applicable for games in the genre of racing games. His biggest worry however, is the restriction of being on a tablet, and touch interface.
One should find the right balance between "control and features", meaning the amount of features available, and how much freedom the level designers have.

- *Do you use any tools for level design?*

He uses primarily paper and whiteboard as a means of working. Playing the level is crucial, in order to judge the quality of a level, and enhance the prototype. Using Lego is clever due to its physical delimitations, allowing everything to have a more or less accurate scale.
He does not use computer tools (he has only 1 year of experience as a level designer).

- *How do you draw your initial level on paper?*

He draws many different routes and tries to find things that are unique for every level. He finds that core idea for a level, and works from there. He places the start and end of a level very early in the process, as his first step.

- *What's a normal creative process of level design like*

He is very chaotic in his way of working. It goes very back and forth. He tries to find new concepts. It is very unorganised in the start, and becomes more and more concrete as you go along.
The beginning can be very overwhelming, and to relate to mechanics can be chaotic.

- How do you tweak a level?

You start with the game concept before doing level design. He keeps focus on: what do I want to teach the player?
(On the game he is working on) He wants to teach him to hurry up. Show that you get more points for killing penguins, introduce challenges.
The role of the theoretical level designer: combine challenges and mechanics.
In the former game he worked on blueprint layout of the hotel.
He used gameglobe, a 3D level design tool. It allowed him to model landscape, it had different mechanics, and to visualize 3D rooms.

- *Once the creative process is done, what are the next steps for you?*

Testing is a great part of the creative process, and after. You have to play it yourself (and make others play), and consider: does it feel right? This is the main reason for the level to take such a long time to make. The feeling of a level and the requirements are essential. He inspires himself of a typical user of the game, and builds a level from there, then come the requirements. You run through a level with the eyes, and make new rapid prototypes and revisit. You do many drawings.
A tool could: collect and show data, for example how long a route takes, how many turns there are, etc...

- *What do you see as the bottleneck in level design?*

Translating a level to the game. The process is very iterative, and when you run into a problem, you fix it straight away. The goal is to have the levels in the final engine as fast as possible.

- *Do you ever have a problem making others understand the vision you have for a level?*

It can be very hard to explain a vision. It is hard to illustrate what you are striving for. Usually you get bad critics from the things that are irrelevant for the process. There is a definite need for illustrating levels.

Transfer an interesting concept to a well designed level. It isn't always the esthetic level, which winds up being the level that works the best. Sometimes you need to destroy the nice structure of a level in order to make it work. One should be able to kill his darlings.
The translating process, is simple if you make it iteratively. You start in what's esthetically nice, and continue on to integrate it more and more, into the context of the game. The iterative process takes time.

When he works, he goes much forward and back, works on a level until he gets bored/uninspired, then jumps on to a different level, until the creativity fades and so on...

- *What is a level editor for you?*
- *What do you expect from a level editor?*

User friendliness, less control. Thankful for the help of programmers. The disadvantage of working with made by programmers, is that you depend on them to do something when you need a feature. Getting a prototype in Unity allows you not to lose so much as you can extend it. It has to be easy to use, easy to test again and again, easy to make changes, reset, save load, responsive, stable, have good visibility, and usability. Being able to manipulate the elements of the game in a fast and satisfying way. Level designers are not unity pros, and should work with it in an easy way. Most importantly, to have everything in one place.

**After testing the tool**

He thinks it would be a shame to make a one size fits all level design tool. It works well being specific to one genre. And a level editor is different for each game. For route and path based games, the tool is nice. For terrain generation as well.
Perhaps one could think greater than just level design. In an adventure game a character could jump over cliffs. Maybe one can save a ghost run. Have a character play the level automatically, controlled by AR. He would really like a PC version of the tool as well. Perhaps on the early stages of level design, it can be interesting to use AR.
It is crucial to keep the user friendliness, and he would really like the possibility to export a level to Unity.

The need for randomly generated heights depends on the project. However in cases it is relevant, it can be interesting to have something that isn't flat that makes sense, for auto generation perhaps.
The tool could be used for both 2D and 3D games, it could cover a lot of other games as well.
The curves are nice, and very interesting to have. However the tiles are simple. The esthetics of the curved road are important. But one should remember that features are more important than esthetics.
He also asked for textures, and being able to set the scene in a different context than just a racer game.
It would be nice to draw a path with your fingers, to select a path, or direct an autoplayer.

The program exceeds the expectations set from the video. There are a lot of features, and they are easy to use. Getting straight to editor mode after generating is good. and shows good usability.
He sees a lot of potential.
Placeholders are fine to work with, but to show what you want to do, he would like the possibility to put in his own objects, fx icebergs.

# Ole Summary

Background: Ole has been working in IO-interactive as a level designers for 5 years. He has won a genre award by creating a board game called “Police Precinct”, inspired by Grand Theft Auto and other murdering/shooting games.

During the interview, he asked some questions for what the tool should be doing as he didn’t get the full overview of what we wanted to achieve. After explaining, he saw positively at its usage, both in small indie games and big game productions. He would not restrict it to small companies.

He uses a lot of other tools like Photoshop, Sketch-up before the physical product. Rendering time and the functional part can take up many minutes. He also uses 3d studio max to get something up and running fast. He works with gameplay and level design simultaneously, also simulating some Artificial Intelligence behaviours along so he can test out the levels quick.

We asked what our tool could help level designers with. He mentioned that our tool could help speed up this process of rendering, executing the parsing and testing of a level. Specs of character is also something he concentrated on, like its speed, visual, the space and its freedom to run around. He mentioned that the space around the character is important as it gives different feelings depending on the environments and environment change. He would like to have basic physic behaviors in the system as well. He talks about the different elements of a level, like a testing dimensions, have different cylinders and objects around the world you can bump into and have other objects such as crates, boxes - where you can hide behind and think how you can use it in the gameplay. He talks about those things giving the flow of a level and to think of the whole levels architecture. He mentions something about meeting enemies in the level also, which gives the player the right impressions.

He gives the example of a shooting game. Then we need room for big battles, maybe some tools like a grenade and placement of other props. Something he mentions, which he thinks our tool would be nice to have, is the ability to sketch something, which translates into enemies, pickups or other objects. Maybe checkpoints for a racer game and then you could measure the time between checkpoints. He mentions that the tool doesn’t necessarily need an interactive environment, but the essential part lies in the visuals.

He moves on by mentioning that game and level design is close to each other and when he works on something, he thinks of having the encounters as in the Wario Ware game, which has quick and specific reactions depending on the situation. He could make a basic level in 3d studio max and import the tool and render it but it takes time. He also mentions having something that breaks the line of sight of the player and control the player behavior in that way. He wants to think where to place the different objects around the scene to give the player different tactics depending on the encounter. For instance, if you are against a big boss, then maybe you have a rocket launcher placed somewhere around the level and it is to find out how and where to place

the rocket launcher in order to make the level enjoyable. He mentions other enemies again and that a red box is enough to visualize that its an enemy.

When he starts to sketch something on a paper, he starts thinking in 3D but draws them in 2D. He mentions having visual cues in a level, leading the player somewhere in a big environment and the idea is to give the player the right directions for where to head towards. Then he user tests how the player navigates. He likes to have visual navigations and when things should appear and direct the player.

He mentions that import and export of a working level would be good. He uses sketchup to drag in objects around and he thinks our tool could be a competitor against it. He works in front of the computer most of the time and sketchup is fine as a tool. He mentions phone and tablet also being a good options as it is very reachable. He works with board games and he says it is like being an author. He thinks of ideas and wants to quickly test them and he thinks that if our tool quickly could scan them, there is a great potential.

We asked him what other tools he uses. He mentioned that big companies have their own tools, for instance IO-interactive. Commercially, he thought of Unity and Unreal Engine. It was a state of the 3D physics world where he could build the world. Importing prefabs build in 3d studio max was also good. Others he could think of was Hammer and Gamemaker.

He continues and mentions the importance of line of sight again, especially in a shooting game. He talks about gating and cutting the world into different sections. Ports also to limit the field of view and also technically less rendering areas to have efficient level renderings.

We asked him how he starts his creative process. He says that he thinks of the story of context first, for instance, a character arriving at beograd metro, and he needs to go from a to b and try to get the idea behind the story. Then he tries to think of the different game elements, weapons and other features. Then he starts paper prototyping and also googling a lot of pictures of beograd metro, try to find ideas, erase many and find new features also. He thinks how to begin an encounter and try to build a prototype from that, get something fast on 3D and tries to get some coding in as soon as possible. He works in an environment with many level designers working simultaneously. After, they try to polish and iterate levels through. Tweak in between and try to find good ideas from other colleagues.

We asked when they test the levels. He answers all the way through and he tries to simulate it as a board game with simple rules and test it as soon as possible. We asked what they do after their creative process. He answers that its mostly about bugfixes, finding test persons to test the levels and basically test it all the way through.

We asked if he had any problems telling others in his team what the vision of a level was. He said that there was a compromise and it was hard to match other peoples visions. But it was not that hard to make others understand your own vision. He says that even though you understand

the game, there are times where misunderstandings happen and the design does not necessarily go in the direction you want it to go. Many things go wrong and it is good to talk about them he says. He says it is good to both talk about the problems and show them the problems visually.

We asked if he uses some standardized symbols for his drawings. He mentions that it is good to have clear symbols and illustrations that express the things you want. In skyrim for instance, you have the helm symbol. In that way, people will have the idea of which direction you are going towards. He makes a sketch of a level, dungeon and dragons like, try to get the concept art. Try to make a gamification of a location. He gives a prison as an example, which gives some restrictions and try to find challenges out from that. Something he thought was hard was to keep things from changing, because of the iteration process from start to end of the production.

He thought usability meant a lot for our tool and to get fast 3D models generated. To navigate around the environment and maybe use it as a playable level for a school class to test out for instance a racetrack. We asked him whether or not an import/export function would be important and he said that it depended on where we wanted to present the level. But he also said that it would be very complex to make a unified export to all kinds of platforms and that we should consider a standard format for specific platforms before considering it. He talks about resources, how many vertices etc, and that an export would not be optimal in such cases for a specific game and that it is all dependant on the game. He sees our tool as a prototype of a prototype and it will most likely get replaced in the end. But he thinks that it is good for rapid prototyping and we just trash the prototype after using it. He says that we need something we believe is working and therefore we support many files but trash most of them. Fast testing and paper is good for that.

We asked him what a level editor was. He said something about manipulation of 3D. A gameplay editor or a world editor. He optimally wanted it to be the place where you play and get the game experience out from it. He focused much on gameplay of a level. In its ground form, it is an abstraction of a world, for instance, mario done as a 2D array. It is the in-between layer which makes it easy to visualize and to interpret the level for other people and to find the right spots. He mentions 3 things that affect a level: Environment, story and its playfulness (how interesting the level is). A level editor should be capable of many things. He mentions Gameglobe, where the editor is the game itself.

Before many tools, he used to play around with tiled papers, number codes and play with papers, blocks etc. He says tiles are fine, but asks if it is possible for a racing game. He says it depends on the market and our target group. It also depends on the platform. Tiling is like pixelation he mentions. As last, he mentions snap-to-grid would be good for the level tool.

After the presentation, he mentions that he would like some triggers, and he sees the product as a demo tool. He says the most important feature is the road generation from a given curve.

# Peter Summary

Primarily car racing games, terrain editor, but hasn't really understood if it's meant as a prototype, or final level. An idea could be to use it as a content creator for people. Developers would usually not use it. Fast drawings make sense. However he has a worry about colliders in unity with mesh collider of terrain.

**Use tools?**
Yes, in different phases of the development. To start with, it's like a playground, where everything is allowed. The idea is to make up something great. To start with, you get information and requirements of the world (the player has a game…), then you change some things, and come in a phase where you play around with different things, in order to find out what works, and what doesn't. After having settled some levels, he builds them in Unity, and does initial tests. He builds puzzles in small rooms, tests himself and gets colleagues to test. Then judges the levels after getting comments, finding what's cool, uses it and scraps what's not.
When drawing by hand, he draws stickmen, basic illustrations, boxes, pressure plates, elevators, doors. He sits on a couch, and usually finds 12 ideas in an hour, and discards many ideas in the process. Then he looks at those 12 ideas, judges them, then build those who have potential. Then he gets into Unity, and tries to build something as quickly as possible. After he makes some considerations about sizes relations, the weight of different things, what you have to do in the level. At this stage they test with their main character, and he interacts with the world. They have some rules and conditions for him. In this case he connects to everything: he can push, crawl, shoot etc. Then he puts the good idea in one box, the bad ideas in another, and the ideas that need work in a third. After they use these levels to iterate on.
Everyone works differently. Some draw their levels on paper in detail, other sketch it, others don't at all. However everyone take notes on blocks of paper in case their is something.

**When drawing**
He uses symbols, but draws only for his own understanding. Therefore it is very messy and no real symbol as to what is what. For example he would draw an elevator as an arrow and a dotted platform. He draws it nicely in photoshop or paint if he were to present it to others. A level is like notes taken by a journalist, only he understands it completely.

**Creative process:**
First he gets information from the instructor as to how the scene is set. For example in a forest where the character is getting chased. However he will ask questions, in order to get a deeper understanding of what is meant, and perhaps get keywords, that he can work from. Perhaps like what you want the player to feel, should he hurry up, be paranoid, relieved? Setting the scene is very important, for example if the level is in the woods, or in a lab. Some things are specific to each scene and need to be considered.
Sometimes they iterate over a mechanic: make up something about gravity. Then it's about finding limitations and opportunities. They use the physics in the Unity engine, so they get a lot of

things for free. Also it is very intuitive, people live with gravity, so there's a lot of things that do not need explaining (things sink or float in water, things fall…). Or if they try to find something out of wind. What can you do from there? They would try all kinds of things, and scrap the bad.

In the early process of Limbo, he wasn't a part of it. They drew the boy on a napkin, and tried to create a universe from there. To start with in a process, they know some things and try other things out. If you make wrong decisions you go back to the basics and start over.
On their new game, they use a 3D engine for a 2D game. So they considered the challenges and opportunities. So they experiment with things, for example with the camera. There's quite a big degree of freedom, and it can be hard to be a level designer, as your job description is to make something fun, and immersive, but the design aspect of gaming, gets very well defined the more you work with it.
*(Context of our tool)* An easy way of generating terrain, and easy way to get the character into the environment and test. It has to be easy to export and to use in unity. It has to be a fast tool, and easy to use. *(He mentions building boxes with weight and elevators)* A feature could be to get things out on paper again afterwards. Maybe to pitch a game idea. Think backwards perhaps.

**After the creative proces**
After drawing levels, he works on Unity. He starts by making a floor, and a killer object, which kills the boy when he touches it. Then looks at what has to be in the level, and the mechanics that work. If you press on the button, you half the gravity. He makes scripts (prototype), tweaks and puts it together so it works. Then he puts the character in, and plays with him. He tries to build all kinds of things, and find out what is fun *(Experience from limbo:* There needs to be ropes and ladders in a 2D world). By now he will start to discard the 12 ideas, and keep on building trying to find something good. He sorts them in an order, in order to optimize work, however you have to remember to enjoy the level while working with it. He constructs the ideas very fast, in order to confirm or discard them. Usually about two ideas make it this far, and get shown to the colleagues. *Limbo:* They worked without deadlocks, so a level could be tweaked any day.

**How do you tweak a level?**
He doesn't do it the same way as a friend in ubisoft. They build the final level from the beginning, and tweak everything about it later: how do the AI move/behave, where should you aim etc… In Playdead they build levels, put them in boxes and iterate over them. They work on mechanics, pulling things, where are things compared to the others, pressure plates, gravity and see what works. They don't think so much about how the pressure plate gets activated.
They test the levels a lot internally, but also externally. You get corrupted when working too much on the game, and automatically assumes too much. Therefore it is important to get it tested externally with new eyes, and people who think differently. It is not important to get them to comment, as it is clear in the way they play if they are struggling or not, and what they are trying to do. For example if the player cannot find a lever, maybe they should make it red etc…
In order to get a feature they make very simple solutions (not pretty) that work, and can prove the idea at hand.

Requirements about the levels. Project manager wishes all levels have a specific length, however every level differ from the player who plays it.

**What takes up most time?**
To iterate over a level. It is quite easy and fast to find ideas, but to get something worthy of being played, and to find what is uncool, is what you use a lot of time on. It is not easy to find out what to cut.

**Talking about tools**
A visual programming tool the animator used, allowed him to make a really good feature the programmers can program.

**About our tool**
Maybe we should focus on a specific genre, and make it right for that one. Create a fictional company which would have great use of the tool.

# Mikkel Summary

Mikkels first thoughts about the project was that we should think more broad than just a generation of a road. He says that our challenge lies in how fast we can generate things. He mentions it looks like a terrain editor he used before in another game, but it was more detailed levels with heightmaps. He says if you move from 2D to 3D, you acknowledge some things. He mentions that it can be hard sometimes relating to the space you are in when you draw and you sometimes figure out that your sketch does not work, in the process of translating from 2D to 3D. He says that you adjust the levels fast and iterative. He works a lot in the creativity process. When he saw the video, he thought it was magic, very simple to make it happen. You draw and you get a world quick and it is very entertaining. He was not sure if it was only meant for a racing game or something else also.

He saw the potential in the idea that you can visualize something which you could not see before. It was an inspirational tool for him, rather than a working tool. He says that the designer cannot always get the same view from others. He tries to find ground ideas for a level and our tool could help him do that. He says that the tool needs to set some premises. He mentions randomness in the inspirations, for instance tunnels, bridges and other stuff could be used in our tool. Also he says we can maybe have tweakable definitions. He would like fancy graphics, lighting, fog and other effects.

He mentions his own project with Max and the Magic Marker. He says he could not see himself use a tool like this for that purpose. He mentions that if we make a game like Tomb Raider, then we would want to have walls to climb up on, some hills, and how high the character can jump. He says that the tool should help making a ground plan and get variation and randomness in the gameplay. He says that we should be specific in what we want to use the tool for. He rounds off with these arguments with: “Can you hop on the tool and get a gaming experience?”.

Afterwards, we ask him about how he draws. He says that he draws fast and try to build it as quick as possible. He says that in the moment where it is in full 3D, something else happens. A lot of proportions change in the space and you are in a different game world.

We ask him how he starts his creative process. He starts off with an example from a shooter game. Then he asks himself the questions, what are we going to do in this level, what is the purpose. Is it to shoot a person, steal a car and take off afterwards? He would start to plot things in the paper, making dots and symbols on persons to kill, something that is escaping etc. He tries to find ideas on how to get in somewhere, what is an interesting way to get in, try to get some ground ideas and see the essence in the things he does. He starts by defining some basic things. He would try to structure around some ground rules. He mentions his Max game again, where the goal was to make every level have a unique mechanic so when people talked about the game, they would be able to specify the level by its unique thing. He tries to set some

boundaries, see what kind of experience the player needs to have (sneaking, shooting, driving etc), make some traps and increase difficulty.

We asked him how he tests. He would first of all play the level a lot himself. Then he would try to test it out with colleagues. Already there, he would catch a lot of bugs. Lastly, we would test it with others outside from the building. He says they test until they reach a quality which is satisfying.

We asked him about the creative process. He mentions that there are many phases for a level designers. First the design of the whole world and gameplay, and try to fit the world that creates the gameplay. He says that there lies a lot of work to get all these settled. For instance, if there is rush hour in a level, then there would be a lot of configuration work to do. Then there are things like finding errors, and fixing errors, which can take a lot of time. He mentions that the fun part is only ⅓ of the process.

We asked him what the bottleneck of the level design process was. He says that if it is time-wise, it is to make the mock of the world, with simple boxes and then make the gameplay good. Setting up the puzzles is easy but the moment where graphics enter the scene, there are lots of other configurations and many things change because of the new graphics elements. He says that it is not easy to read other peoples thoughts. So the process goes from being a simple level to a very detailed world.

We ask him what a level editor is. He says Unity is a direct level editor tool. But he also says that he worked with other level designers using sketchup to make more complex models, but there are many things not possible in that tool. Sketchup is easy and precise. He says that historically, there are two types of level designers. One which focuses on gameplay and the other for the architecture and form of the level.

Appendix B

# Test Results

We had many different test cases to test our environment. Some of the cases are already mentioned in the Test section. Here we will show their resulting environments.

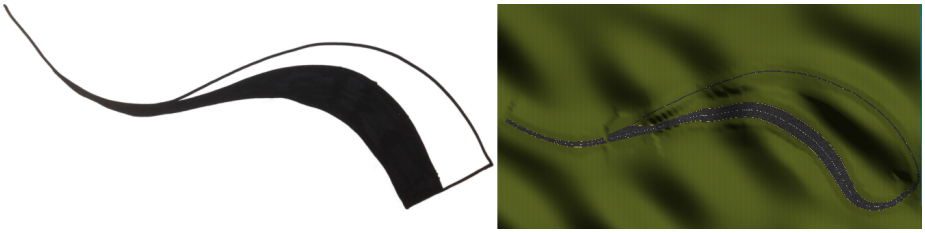


**Figure B.1:** Compare the "Logo" test.

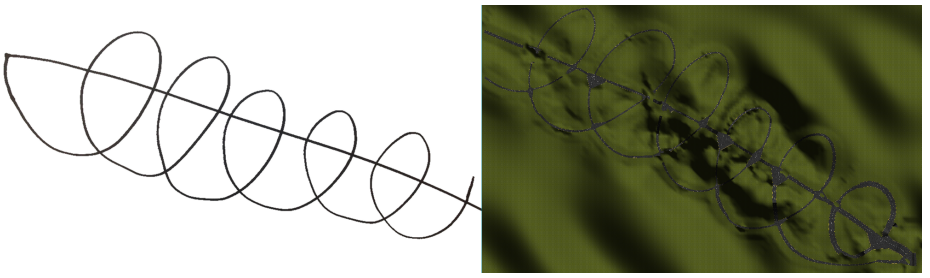


**Figure B.2:** Compare the "Swirl" test.

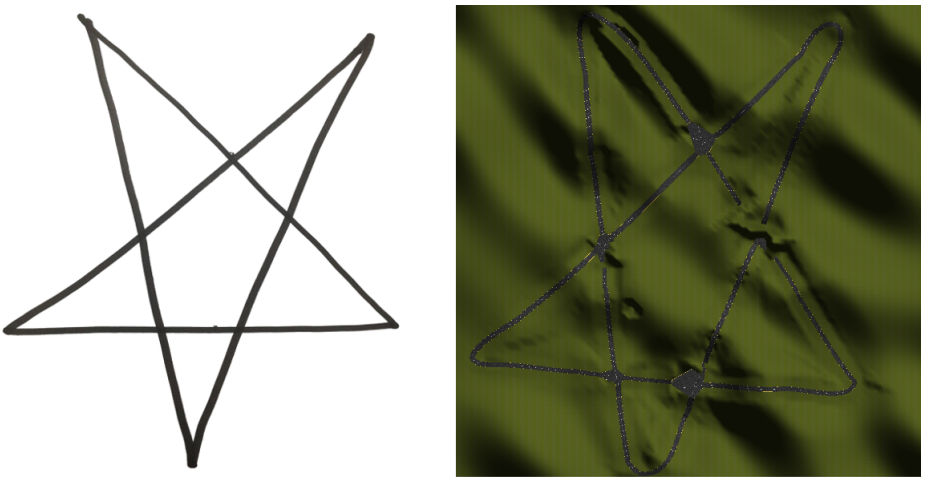


**Figure B.3:** Compare the "Star" test.

Appendix C

# Least Square Implementation

The implementation is done in C#.

**Listing C.1:** LeastSquare.cs

```
public Vector2[] BestFit (List<Vector2> points)
{
  Matrix M = new Matrix(4,4);
  M[0,0] = -1; M[0,1] =  3; M[0,2] = -3; M[0,3] =  1;
  M[1,0] =  3; M[1,1] = -6; M[1,2] =  3; M[1,3] =  0;
  M[2,0] = -3; M[2,1] =  3; M[2,2] =  0; M[2,3] =  0;
  M[3,0] =  1; M[3,1] =  0; M[3,2] =  0; M[3,3] =  0;

  Matrix Minv = new Matrix(4,4);

  if ( (int) MatrixDeterminant(M) == 0)
  {
    Minv = MatrixInverseSPD (M);
  }
  else
  {
    Minv = MatrixInverse(M);
  }

  float[] normalizedPathLengths = NormalizedPathLengths(points);

  Matrix U = new Matrix(points.Count, 4);

  for (int i = 0; i < normalizedPathLengths.Length; i++)
  {
```

```
    U[i,0] = Mathf.Pow(normalizedPathLengths[i],3);
    U[i,1] = Mathf.Pow(normalizedPathLengths[i],2);
    U[i,2] = Mathf.Pow(normalizedPathLengths[i],1);
    U[i,3] = Mathf.Pow(normalizedPathLengths[i],0);
  }

  Matrix UT = MatrixTranspose(U);

  Matrix X  = new Matrix(points.Count, 4);
  for (int i = 0; i < points.Count; i++)
  {
    X[i,0] = points[i].x;
  }
  Matrix Y  = new Matrix(points.Count, 4);
  for (int i = 0; i < points.Count; i++)
  {
    Y[i,0] = points[i].y;
  }

  Matrix A = MatrixMultiplication(UT, U);
  Matrix B = Matrix.Identity(4);

  if ((int) MatrixDeterminant(A) == 0)
  {
    B = MatrixInverseSPD (A);
  }
  else
  {
    B = MatrixInverse(A);
  }

  Matrix C = MatrixMultiplication(Minv, B);
  Matrix D = MatrixMultiplication(C, UT);
  Matrix E = MatrixMultiplication(D, X);
  Matrix F = MatrixMultiplication(D, Y);

  Vector2[] finalPoints = new Vector2[4];
  for (int i = 0; i < 4; i++)
  {
    float x = E[i, 0];
    float y = F[i, 0];

    finalPoints[i] = new Vector2(x,y);
  }

  return finalPoints;
}
```

# Bibliography


[AfARA05] Abstraction and Implementation Strategies for Augmented Reality Authoring. Vuforia smart terrain, 2005.

[API14a] Unity3D Scripting API. Mesh. http://docs.unity3d.com/ScriptReference/Mesh.html, 2014.

[API14b] Unity3D Scripting API. Webcamtexture. http://docs.unity3d.com/ScriptReference/WebCamTexture.html, 2014.

[Cha14] Goose Chase. Goose chase. https://www.goosechase.com/, 2014.

[Dou14] Andrew Doull. Procedural content generation wiki. http://pcg.wikidot.com/, 2014.

[Gla13] Martin Glaude. Procedural mesh generation [simcity roads]. https://www.youtube.com/watch?v=dDqCPMpX1vI, 2013.

[Goo13] Kim Goossens. Procedural road creation. https://cmivfx.com/store/213-houdini+procedural+road+creation, 2013.

[Her12] Jim Herold. Least squares bezier fit. http://jimherold.com/2012/04/20/least-squares-bezier-fit/, 2012.

[How14] HowToGeek. Which video game was the first to feature procedural generation. http://www.howtogeek.com/trivia/which-video-game-was-the-first-to-feature-procedural-generation/, 2014.

[JDG13] Éric Guérin Adrien Peytavie Bedřich Beneš Jean-David Génevaux, Éric Galin. Terrain generation using procedural models based on

hydrology. Technical report, Université de Lyon, Laboratoire LIRIS, France, Purdue University, USA, 2013.

[Jon13] Peter Jones. Infinite procedurally generated 3d terrain in unity. https://www.youtube.com/watch?v=lUi_k1Qqkh0, 2013.

[JT11] Rafael Bidarra Julian Togelius, Jim Whitehead. Guest editorial: Procedural content generation in games. http://graphics.tudelft.nl/~rafa/myPapers/bidarra.TCIAIG.2011c.pdf, 2011.

[Jø12] Bilal Arslan & Patrick Jørgensen. Car Simulation for Neurofeedback Research. Technical report, Technical University of Denmark, DTU Informatics, 2012.

[lev12] World of Level Design. Technical report, 2012.

[Mac98] Wendy E. Mackay. Augmented reality: Linking real and virtual worlds a new paradigm for interacting with computers, 1998.

[MM10] Paul Merell and Dinesh Manocha. Model synthesis: A general procedural modeling algorithm. http://graphics.stanford.edu/~pmerrell/tvcg.pdf, 2010.

[Moj09] Mojang. Minecraft. https://minecraft.net/, 2009.

[Ngu12] Thierry Nguyen. Clash of the dotas. https://http://www.1up.com/features/clash-dotas-league-legends-heroes, 2012.

[Ols04] Jacob Olsen. Realtime procedural terrain generation, 2004.

[Qua12] Qualcomm. Image targets in unity tutorial. https://www.youtube.com/watch?v=WrEnREOT1F0, 2012.

[Qua13] Qualcomm. Vuforia smart terrain. http://www.youtube.com/watch?v=UOfN1plW_Hw, 2013.

[RD12] Jose Luiz de Souza Filho Rodrigo Silva Marcelo B. Vieira Bruno Dembogurski Renan Dembogurski, Dhiego O. Sad. Interactive virtual terrain generation using augmented reality markers, 2012.

[Rez13] Nayef Reza. Zhang-suen thinning algorithm. http://nayefreza.wordpress.com/2013/05/11/zhang-suen-thinning-algorithm-java-implementation/, 2013.

[RFW03] A. Walker R. Fisher, S. Perkins and E. Wolfart. Thinning. http://homepages.inf.ed.ac.uk/rbf/HIPR2/thin.htm, 2003.

[RS10] Klaas Jan de Kraker Rafael Bidarra Ruben Smelik, Tim Tutenel. Integrating procedural generation and manual editing of virtual worlds. https://blog.itu.dk/mpgg-e2010/files/2010/10/a5-smelik.pdf, 2010.

[Sur13] Jayelinda Suridge. Modelling by numbers. http://jayelinda.com/modelling-by-numbers-part-1a/, 2013.

[Uni14] Unity. Generating mesh geometry procedurally. http://docs.unity3d.com/Manual/GeneratingMeshGeometryProcedurally.html, 2014.

[You09] Shamus Young. Procedural city. http://www.shamusyoung.com/twentysidedtale/?p=2940/, 2009.