Bachelor Project:

RealMaps: Using real maps in Minecraft

Authors: Sindre Helleborg
Tellef Møllerup Åmdal

Date: 15.05.2015

Table of Contents

1. Abstract . . . 3
2. Sammendrag . . . 3
3. Preface . . . 4
4. Introduction . . . 5
   4.1 Project Goal . . . 6
   4.2 Earlier Work . . . 6
   4.3 Team Background . . . 6
   4.4 Report details . . . 7
      4.4.1 High Dynamic Range . . . 7
      4.4.2 Level Set Methods . . . 8
      4.4.3 Minecraft . . . 8
      4.4.4 Terminology . . . 10
5. Methodology . . . 11
   5.1 Software Development Process . . . 12
   5.2 Design . . . 12
   5.3 Program Specifications . . . 13
   5.4 RealTerrain . . . 18
      5.4.1 Biomes Selection . . . 19
      5.4.2 Bottoms up Midpoint Displacement Algorithm . . . 20
      5.4.3 HDR Compression Applied to Terrain . . . 21
      5.4.4 Level Set Noise Reduction . . . 27
      5.4.5 Map Analysis Algorithm . . . 31
      5.4.6 Open street map feature parser. . . . 32
   5.5 RealMapsWorldType . . . 33
      5.5.1 Configuration . . . 34
      5.5.2 Generator . . . 35
      5.5.3 Other uses . . . 36
   5.6 Website . . . 36
   5.7 User Testing . . . 37
6. Result . . . 38
   6.1 User Tests . . . 39
7. Conclusion . . . 41
8. References . . . 43
9. Appendix . . . 44
   9.1 Original design notes, Ben Sawyer . . . 45
   9.2 Initial thoughts and ideas . . . 47
   9.3 Meeting notes bundle . . . 47

1. Abstract

Title	RealMaps
Date	15.05.2015
Participants	Sindre Helleborg
	Tellef Møllerup Åmdal
Supervisor	Simon McCallum
Employer	Ben Sawyer, Digimill
Contact Person	Simon McCallum, simon.mccallum@hig.no, 61135268
Keywords	Minecraft, Maps, Terrain, Geographical data, Geography
Pages	52
Attachments	3
Availability	Open
Abstract	The project was to create an automated system that would take publicly available geographic data and use them to create to playable Minecraft worlds. These maps where to be accessible through a web server running our system. The focus of the project was the technical simplicity to the end user and the similarity between our maps and those generated by an ordinary minecraft version, both aesthetically and gameplay wise. Our solution ended up being 2 different java applications working in a pipeline. We created these two and a proof of concept website made with django. The first program handles downloading the relevant geographical data and processing it into a form that can be used to directly converted to a Minecraft map. The second one being a Minecraft mod, used for generating the actual Minecraft world. It uses Minecrafts own built in generator with our code running on top. We also did some user tests. We let people navigate through the generated Minecraft worlds. We gave them a map of a real world location and tasked them with finding the corresponding location in Minecraft. We also did the opposite where we had them try and mark a location on the map based on a location in Minecraft.

2. Sammendrag

Tittel:	RealMaps
Dato:	15.05.2015
Deltakere:	Sindre Helleborg
	Tellef Møllerup Åmdal
Veiledere	Simon McCallum
Oppdragsgiver	Ben Sawyer, Digimill
Kontaktperson	Simon McCallum, simon.mccallum@hig.no, 61135268
Nøkkelord	Minecraft, Maps, Terrain, Geographical data, Geography
Antall sider	52
Antall vedlegg	3
Tilgjengelighet	Åpen
Sammendrag	Prosjektet gikk ut på å lage et automatisk system som tar offentlig tilgjengelig data og bruker det til å lage en spillbar Minecraft verden. Disse kartene skal være tilgjengelige igjennom en web side som kjører systemet vårt. Hovedmålene med prosjektet var å lage et system som ikke krevde store tekniske ferdigheter av sluttbrukeren og at kartene laget av systemet ligner på de i en vanlig versjon av Minecraft, både når det gjelder estetikk og spillbarhet. Løsningen vår var to Java applikasjoner som arbeider serielt, vi lagde disse to og en test webserver lagd med Django. Det første programmet håndterer nedlasting av relevant geografisk data og bearbeider det til en form som kan bli direkte konvertert til et Minecraft kart. Den andre er en Minecraft mod brukt til å generere det faktiske Minecraft. kartet Den bruker Minecrafts innebygde generator med vår kode kjørende på toppen. I tillegg gjorde vi noen bruker tester. Vi lot folk navigere de genererte Minecraft verdene. Vi ga dem et kart over et område i den virkelige verden og ga dem oppgaven med å finne den tilsvarende posisjonen i Minecraft. Vi gjorde også det motsatte, der oppgaven vi ga dem var om å finne en posisjon på kartet markert i Minecraft.

3. Preface

The Minecraft RealMaps project was the brainchild of Ben Sawyer. As game programming students we wanted a game related project and saw this project as a perfect fit, as both have a quite extensive experience with Minecraft. We also saw it as an interesting technical challenge creating a complete working service requiring several interdependent parts. Working on creating something for an already existing game meant that it would be easier to complete something a usable end product, that is something more than a tech demo.

We would like to thank Ben Sawyer and Simon McCallum for their guidance during this project.

4. Introduction

Project Goal

> *Goal : Make it possible to create HIGHLY playable maps from real geographies using publicly accessible geographic data enabling new types of play and learning inside Minecraft.*

Ben Sawyer
— *Appendix A*

The task is to create an automated system for generating Minecraft maps from real world geographic data. The main focus is that the map preserves gameplay and the aesthetics of ordinary Minecraft maps while still recognizably representing a real world location. It is important that the customer facing system be simple enough that even people without the experience to modify Minecraft can use it, so it is essential that the maps delivered can be played on an unmodified Minecraft version.

Our target demographic consists two groups, people who play Minecraft and people who would want to use a Minecraft map of a real area for some other purpose. For example: a teacher might want to use a Minecraft map of an area to teach pupils about about the history of that area. Another example is if a city council wanted to display their plans for a certain area in an accessible way. The common denominator with these groups is that we cannot make assumptions about their level of technical skill in regards to Minecraft.

Earlier Work

There have been some attempts to create Minecraft map generators based on real world data like ChunkMapper[15]. Generally the problem with them have been a focus on accuracy rather than a focus on playable maps. We do not know of any examples that generates the underground the same way as in Minecraft. Another thing that was a bit lacking was the generated vegetation being a bit lackluster.

What our project will try to do differently is to preserve the Minecraft gameplay and aesthetics.

Team Background

Our team consists of Sindre Helleborg and Tellef Møllerup Åmdal. We both are doing our 3rd year on the Game Programming line at Gjøvik University College (GUC). Both of us have played Minecraft since its alpha stage.

Sindre has a completed a few semester of a master in physics, the extra math has been helpful for this project. He did not really do much programming before starting his Bachelors degree.

Tellef has previous experience modding for Minecraft and with Java. His history with programming starts in Upper secondary school VG2 with learning AS3 in the IT2 course and afterwards taught himself Java. Following Upper secondary school he went on to taking Game Programming at GUC.

From the Game Programming line, we both learned C++, Java, Python, some SQL and a bit of PHP.

Report details

This thesis is separated in to 3 distinct parts and the appendix. The first being the introduction followed bu what we did and ending with our results and conclusion

Following this section there is a collection of explanation of things one should know when reading this thesis. After that is the next part, beginning with how we organized us and with what. Following that is explanation of the 3 major software components made by us. Capping up this part of the thesis is an explanation of how our testing was done. Starting up on third part of this thesis is the actual results from user tests followed by our conclusions, references (end-note) and appendix.

High Dynamic Range

The human eye can can capture light intensity at dynamic range of five orders of magnitude simultaneously[6]. By comparison a modern LCD screen can display a relative difference in light intensity by a factor of a few hundred or about two orders of magnitude dynamic range. The problem is that in a scene with bright and dark areas, a human viewing an ordinary photograph of that scene may not be able to distinguish details that he or she would have been able to if present when the photograph had been taken. Meaning that photographs are not as representative of areas as they could be. While there are cameras and techniques that can capture this range there are no conventional displays capable of matching the feat. This makes raw HDR image data useless for humans, linear compression would result in an image almost exactly the same as if you had just used ordinary photography. What is needed is a way to compress the range down into something that can be displayed on conventional media without removing details from the image.

HDR compression algorithms takes high dynamic range data and compresses them into low dynamic range data without losing detail, and in such a way that the resulting dataset

still looks like the scene it represents. There are several algorithms that do this, most of them rely on the assumption that humans are more sensitive to local differences to global ones[4].

Level Set Methods

Level set methods is a conceptual framework used to solve various problems numerically, particularly those involving changing topology. Working on curves that split or merge will in most circumstances require a lot of considerations for special cases. Another way of handling these are seeing them as an intersection between a three dimensional surface and a plane. For example a two dimensional scalar field with the scalar value representing height. If the scalar field represents the probability that a part of an image represents a feature, the intersection with the plane might represent the probable edge of the feature. Evolving the scalar field would then be edge detection. This way there would be no need for handling splitting if the edge encompassed several features. In addition doing it this way allows us to solve fairly complex problems on a simple Cartesian grid. If more information about this topic is desired there are many resources available online[16].

Minecraft

What it is

From the Minecraft wiki front page:

> MINECRAFT *is a sandbox construction game created by Mojang AB founder Markus Persson, and inspired by the* Infiniminer, Dwarf Fortress *and* Dungeon Keeper *games. Gameplay involves players interacting with the game world by placing and breaking various types of blocks in a three-dimensional environment. In this environment, players can build creative structures, creations, and artwork on multiplayer servers and singleplayer worlds across multiple game modes.*

07.05.2015
— *http://minecraft.gamepedia.com/Minecraft_Wiki*

It is a highly popular game played by many. Minecraft has in fact sold over 60 million copies across multiple platforms including PC, Android, iOS, XBox 360, XBox One, PS3 and PS4.

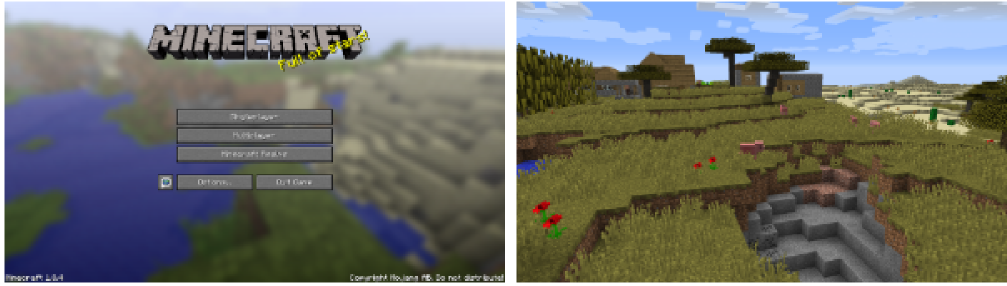

Figure 4.1: Main menu **Figure 4.2:** Savannah and desert biome plus a village.

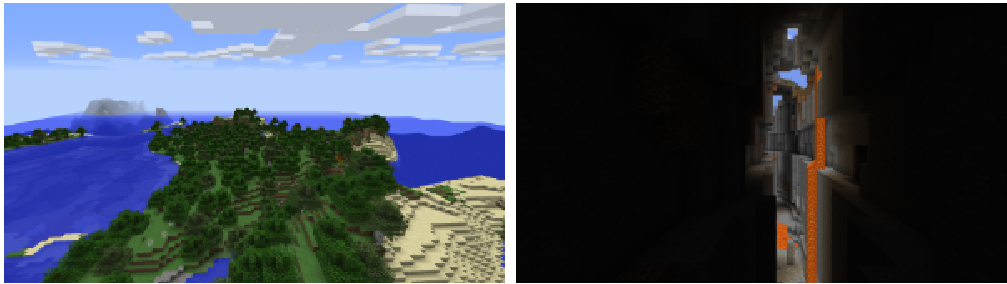

Figure 4.3: An oak forest biome on an island **Figure 4.4:** A ravine with lava.

Most things in Minecraft is blocks, block based or pixel sprites. A block in Minecraft is typically 1m x 1m x 1m size, a $1m^3$ cubical volume. When one plays Minecraft there is generally the choice between survival or creative mode but that alone is not always the sole deciding factor of how tings are played. The greatest reason behind that is custom maps and server side scripts creating a different game experience with it.

Creative mode gives the player to shape the world however they pleases. The player has everything they could use at their disposal and has the ability to fly. Infinite blocks and instant block mining is convenient for creative work.

Survival, as the name implies, is a mode where surviving is part of gameplay. Things like hunger, health, the environment and hostile creatures are some of the things a player has to consider at all times. Mining blocks takes time and appropriate tools has to be used to be effective and efficient.

The game world consists of blocks representing different kinds of terrain, trees, villages, oceans and so on. Even tho one could say the world is infinite in the horizontal plane, one can *only* go 30'000km[8] from the centre before hitting an invisible wall. In comparison, the possible map surface is 7 times greater than the earth's surface. So, realistically for players, the world goes on forever.

Before a world is generated, a generator and its settings has to be selected. The default being a generator that uses a seed and multiple algorithms for the different parts it has to generate. The default world generation is done in multiple passes, where each pass builds on the previous one. Generally, the first thing is that a foundation is created with only 3

kins of blocks. Those are: air, water and stone. Air denotes empty space, water denotes where oceans should be and stone denotes the general shape of the terrain. By how Minecraft works, all positions not occupied by a block is an air block, but its more like a 0 data value. At the same time, the biome distribution is generated according to the seed and their own placement rules, for example no desert besides a cold biome etc. Then, the blocks are modified according to what biome they are in. After that, the world is populated with things like grass, trees, villages and so on. Somewhere in there, the underground cave systems and ravines are generated.

Modding

A mod is a collection of media, mostly but not limited to code, images, sound, models and plain text. When used in conjunction with the application it was made for leads to some sort of change. Modding is generally the act of creating, modifying or maintaining a mod.

In relation to Minecraft, modding could lead to all sorts of changes in-game. Everything from flying pigs, a new type of flower, deserts everywhere or just a modification to the graphical user interface like moving the health meter.

Forge/FML (as of Q1 2015) is one of the most used modding framework for Minecraft. FML, or ForgeModLoader as it stands for, is the library responsible for loading, initializing and managing mods. Forge is the framework where its purpose is to act as a compatibility layer between mods and Minecraft itself, handling most (if not all) common additions and alterations.

Terminology

Biome - Unless otherwise specified refers to real world biomes. A contiguous area with similar climactic conditions, typically with with similar types of vegetation

Biome - (*Minecraft*) Regions in the world with similar characteristics is the same biome. The biome value is per vertical column of blocks.

Block - (*Minecraft*) A singular cubical volume. Could also be said to be a voxel. A piece of "Dirt" or "Oak Wood" is a **Block** when placed in the world.

Chunk - (*Minecraft*) An area consisting of 16 Sections vertically. Often the section part is omitted and instead referencing the blocks directly. Grid aligned on 16 blocks on both x and z axis.

DEM - Digital Elevation Model. A digital model of a terrain surface.

Entity - (*Minecraft*) A dynamic object in the world, for example the player or an item on the ground.

FML - (*Minecraft*) The framework responsible for loading mods for Minecraft.

Forge - (*Minecraft*) The framework responsible for exposing Minecraft's internal systems to mods in a way that it avoids conflicts and instability. It also provides useful functionality to make certain common tasks easier.

Forge/FML - (*Minecraft*) Collective reference to **Forge** and **FML**. Seldom does one use one without the other.

Generator - (*Minecraft*) A piece of code that is uses to generate the **World** with. Some generators has options that can be tweaked to the players content before the world is generated.

HDR - High Dynamic Range

Item - (*Minecraft*) A Singular object in the players inventory or in the players world. For example a "Fishing rod" or "Iron Ingot", note that **Blocks** when picket up is represented as items.

Minecraft - A highly popular construction and / or survival based multiplayer and singleplayer sandbox game where almost everything is cubes.

Mod - A collection of media, mostly but not limited to code, images, sound, models and plain text. Its purpose to be used with a game to modify it.

Overworld - (*Minecraft*) The default dimension in a Minecraft world.

Region - (*Minecraft*) An area spanning 32x32 Chunks horizontally. Grid aligned on 32 **Chunks** on both x and z axis.

Section - (*Minecraft*) An area of 16x16x16 **Blocks**. Grid aligned on 16 **Blocks** on x, y and z axis.

World - (*Minecraft*) The world a player interacts with and is playing in. It consists of **Bl ocks** on a grid divided into **Sections**, **Chunks** and **Regions**. The form of the land is decided by the generator selected and the seed it uses.

WorldType - (*Minecraft*) The type of world to generate, is one of the things a player can adjust when adjusting generator settings.

5. Methodology

Software Development Process

At the beginning of this project we selected a scrum like model with development sprints as short as one week. Planning was handled as weekly Skype meetings, where we would discuss the needs of the project and take on assignments. Decision making was purely by consent. The reasons for choosing a development process with so little hierarchy and detailed long term planning was the fact that we only had a team of two developers and that even though we structured the end result as a pipeline, each part was relatively independent. There were no requirements for any off the components be finished before development could begin on any of the others ones and each part missing could easily be emulated, so development of each part could happen simultaneously. In addition both of us have worked together before without running into major problems. The place we foresaw potential problems where the interface and data structure connecting the different components, to avoid getting into trouble here we planned out the format of data sent between applications early and made sure to discuss it before making any changes to these formats.

As part of this process we used a JIRA for issue tracking and sprint management. This let us organize and divide up tasks better between the 2 of us. As for organizing information like format definitions and project structures we use Confluence. We also used it to write this project thesis on as a matter of fact.

When it comes to staying organized with our codebase we utilized git, as any sane developer has make use of source control solutions. Our git repositories is hosted on bitbucket.org as it was the most convenient solution. Another thing we used to help development was a CI (Continuous Integration) setup for build automation. In our case it was Jenkins. The main reason we set one up is so that we had access to the latest version of the code in compiled form, making it generally easier to test and use with other modules. We also configured that when the repository on Bitbucket received a push, it would trigger the build system. Jenkins would then pull changes for said repository and build it per configuration.

Design

We decided to structure our project as a pipeline consisting of several programs, defining the format at each stage early while keeping the individual implementation as flexible as possible. We designed the system in such a way as for it to be possible for a customer to place an order on a web page and the system could carry out the operation independent of any further human intervention. The system works by taking configurations from the web page and feeding them into the RealMap Generator, downloads the relevant data from 3rd party servers and generates map data. The System then uses a modified

Minecraft server to generate maps based on the map data. The RealMap generator determines placement of biomes and the height of the terrain but the mod lets the standard Minecraft systems handle the placement of vegetation and ores, in order to keep the look as consistent as possible with ordinary Minecraft maps.

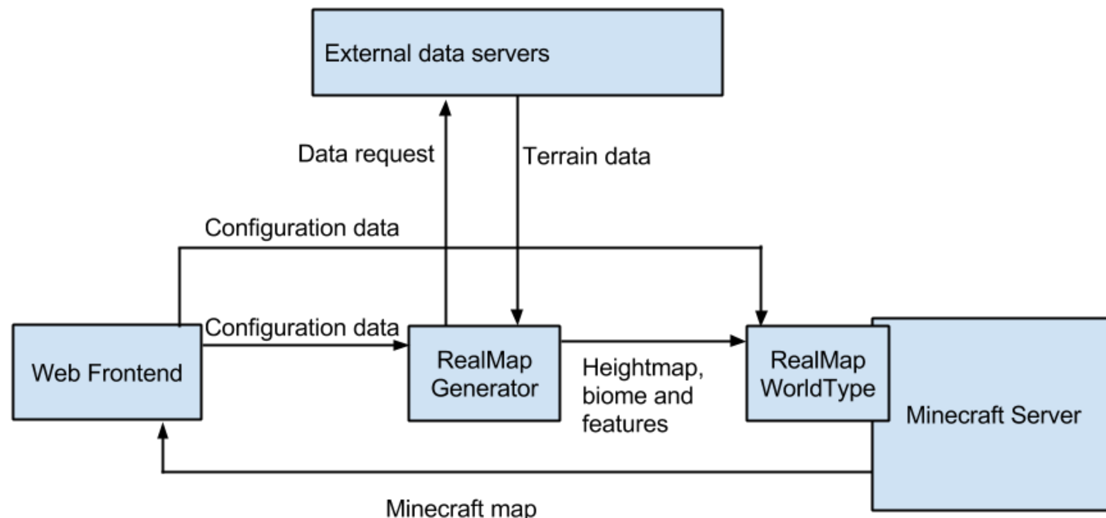

Figure 5.1: Planned modules for the map generator to the website.

When we ended up at the conclusion that interfacing with Minecraft in some way, a decision was made to split the application up into at least 2 parts. The one that interfaced with Minecraft and the ones that did the rest. This was based on the fact that the easiest way to interface with and extend Minecraft is as a mod for the game, specifically a Forge based mod. Also on the fact that Minecraft is quite the heavy burden on the system and downloading, combining and adjusting geographical data don't require Minecraft at all. Splitting up the solution also gives more flexibility to its uses. Lastly there is also another good reason for this divide, licensing code. As modding when it comes to copyright and selling product can be a legal minefield, we had the divide as a way to keep the code bases separate when it comes to licenses.

Program Specifications

Command line arguments for the RealTerrain application

Keyword	Default	Required	Keyword Only	Explanation	Type
-input	-	true	false	What file to read from.	String

-output	-	true	false	What folder to write to.	String
-latitude	-	true	false	Bottom latitude.	Double
-longitude	-	true	false	Left longitude.	Double
-import	asc	false	false	What kind of data importing.	Import
-export	realmaps	false	false	What kind of data exporting	Export
-osmserver	see below*	false	false	What osm server to read from.	String
-temp	temp	false	false	What folder to store temporary data in.	String
-yscale	46.0	false	false	The vertical scale of the generated world.	Double
-xzscale	46.0	false	false	The horizontal scale of the generated world.	Double
-compression	-	true	false	the type of data compression used	Compression
-iterations	100	false	false	The number of iterations for the level set noise remover	Integer

-roughness	0.3	false	false	The roughness of the terrain generated to fill in missing data	Double
-debugfolder	-	false	false	What folder to write debug data to.	String
-debug	false	false	true	Whether to write debug data.	Default
-printmanifest	false	false	true	Print manifest content, then exit.	Default
-help	false	false	true	Print the help text, then exit.	Default

*http://www.overpass-api.de/api/xapi

RealMaps WorldType

Folder Structure

- <name> (Folder)
    - config.json (Main config file) Contains main configuration points, like villages on / off.
    - features.json (Feature list) List of where certain features should be placed, like signposts.
    - data (Folder)
        - region.<x>.<z>.png (Region height-map) Contains all necessary data to generate an entire region.

File content definitions

(JSON) Main config file

```
{
    "Name":"string",
    "Latitude":0.0,
    "Longitude":0.0,
    "Width":0,
    "Height":0,
    "Scale":0.0,
    "Stronghold":true,
    "Villages":true,
    "WorldBorder": {
        "X":0.0,
        "Z":0.0,
        "Size":0
    }
}
```

- "Name" (String) name of map.
- "Latitude" (Double) world position, latitude wise.
- "Longitude" (Double) world position, longitude wise.
- "Width" (Integer) data width in blocks.
- "Height" (Integer) data height in blocks.
- "Scale" (Double) what scale on the xz plane this map is generated with. 2 would mean a 1:2 scale map for block to meter.
- "Stronghold" (Boolean) whether or not there should be strongholds generated.
- "Villages" (Boolean) whether or not there should be villages generated.
- "WorldBorder" (Object) (Optional, can be null)
  - "X" (Double) x position of the center
  - "Z" (Double) z position of the center
  - "size" (Integer) The width and height of the world border

(JSON) Feature list

```
{
    "Signs":[
        {
            "X":0,
            "Y":0,
            "Z":0,
            "Standing":true,
            "Facing":0,
            "Line1":"string",
            "Line2":"string",
            "Line3":"string",
            "Line4":"string"
        }, ...
    ],
    "Beacons":[
        {
            "X":0,
            "Y":0,
            "Z":0,
            "Color":"string"
        }, ...
    ]
}
```

- "Signs" (Array of Object)
  - (Object) represents a sign
    - "X" (Integer) The x position of the sign
    - "Y" (Integer) The y position of the sign
    - "Z" (Integer) The z position of the sign
    - "Standing" (Boolean) Whether or not the sign is standing or wall mounted
    - "Facing" (Integer) The direction the sign is facing, see this page for values to use
    - "Line1" (String) The first line on the sign
    - "Line2" (String) The second line on the sign
    - "Line3" (String) The third line on the sign
    - "Line4" (String) The fourth line on the sign
- "Beacons" (Array of Object)
  - (Object) represents a beacon
    - "X" (Integer)The x position of the beacon
    - "Y" (Integer) The y position of the beacon
    - "Z" (Integer) The z position of the beacon
    - "Color" (String) TODO

(PNG) Region height-map

Name: region.<x>.<z>.png

The x and z is a direct mapping to Minecraft region files.

Size: 512x512

Bit depth: 32 bit

- 8bit: Height-map data
- 8bit: Biome data
- 8bit: Water level data
- 8bit: Flags (from least to most significant bit)
    - Ignore height data
    - Ignore biome data
    - Ignore water data
    - -
    - -
    - -
    - -
    - -

RealTerrain

Language: Java

The language chosen for this part of the project was Java. We believe that Java was the obvious choice, the development team have experience with it and we felt that it offered a good balance between development time and performance. While C++ had been another alternative the longer development time would not have been worth the increased performance, especially because getting that extra performance would have meant even more development time. As this is not a real time system it was just not worth it. In addition the need for the mod to run Java means the project would have required it in any case.

Library: JSON

Since we decided to use JSON as in our data format and the most practical library for this was json.orgs own implementation.

Build system: Maven

As one of our team members had previous experience with Maven, we deciced to use it as our build system for this project.

The RealTerrain processes third party data into a form the Minecraft mod uses to generate maps. The program is started and runs to completion without human intervention. Configuration of the program is handled by command line arguments. The launcher parses the command line arguments and verifies that all the necessary ones are present and valid. If this is not the case the program throws an exception.

The program loads the DEM data from the folder designated in the command line arguments. Currently the only format supported is the ARC/INFO ASCII GRID format[9]. The program is designed to be as independent of the format as possible, so supporting additional formats should pose little problem. The program creates a map stored as a two dimensional array large enough to contain the data based on the scale of the data and desired scale of the end product. Since both the DEM data and the output map is aligned as up being north all we have to do is scale the cell location in the input data to find the location in the output data. The location cell in the input data is scaled to find out which cell in the map it goes into. If several cells scale to the same location the average of them is used in the map. Cells with missing data in them are filled in with the Bottoms Up Midpoints Displacement Algorithm.

At this point additional data is downloaded from an OpenStreetMap server to give additional terrain features such as woods and lakes. The map data is run through the level set noise reduction algorithm to reduce the amount of noise and artifacts in the data. The compression algorithm specified by the command line arguments is then applied to the map data to reduce its dynamic range to something representable by Minecraft. The current alternatives being gradient domain and linear compression. Finally the height data and feature data is combined and exported in the form of a png image together with files in a JSON format with additional map specifications and data.

Biomes Selection

In order to do the most best terrain representation we had to utilize the broadest selection of Minecraft biomes that was practical. We went through Minecrafts different biomes[13] and selected most of those that reasonably represents real world terrain without simply being simply hillier versions of others with the same vegetation. We ended up with selecting the following: ocean, plains, desert, extreme hills, forest, taiga, swampland, ice plains, beach. jungle, cold beach, cold taiga and savanna.

From OpenStreetMap we have extracted the following relevant terrain features: water, wetland, wood, needleleaved wood, broadleaved wood and beaches. Not enough to represent all the biomes.

In order to have more of a basis for selection we divided the world up into six different zones: Arctic, boreal, temperate, tropical desert, tropical savanna and tropical rainforest. The program selects the general biome based on a png image with different specific colours representing different biomes. In order to make the lookup as easy as possible the image was made in a equirectangular projection[14] so that the transformation from latitude and longitude to pixel coordinate is a simple linear transformation.

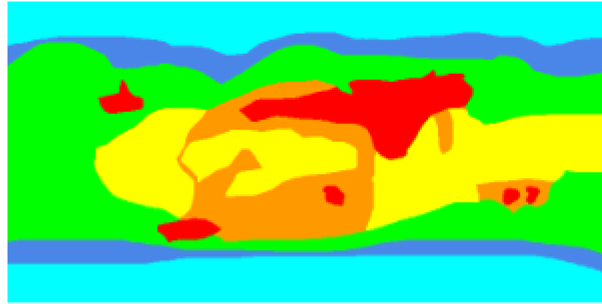

Figure 5.2: The lookup image dividing the world into different biomes.

we combined these two data sources to determine the particular Minecraft biome in the following matrix.

	Arctic	Boreal	Temperate	Tropical		
				Desert	Savanna	Jungle
Not Specified	Ice Plains	Extreme Hills	Plains	Desert	Savanna	Plains
Water	Ocean	Ocean	Ocean	Ocean	Ocean	Ocean
Wetland	Swamp	Swamp	Swamp	Swamp	Swamp	Swamp
Wood	Cold Taiga	Taiga	Forest	Forest	Forest	Jungle
Needleleaved Wood	Cold Taiga	Taiga	Taiga	Taiga	Taiga	Jungle
Broadleaved Wood	Cold Taiga	Forest	Forest	Forest	Forest	Jungle
Beach	Cold Beach	Beach	Beach	Beach	Beach	Beach

Bottoms up Midpoint Displacement Algorithm

The DEM data we download had areas of faulty or missing data, hence the need for an algorithm to fill in the missing pieces. The midpoint displacement or diamond-square algorithm is a good candidate for making plausible terrain. The problem with using it in its canonical form is that since it starts from the edges and generates the terrain top down is that the missing terrain would end up with terrain completely unrelated to the surrounding terrain. A solution to this problem was suggested by Belhadj an we decide to implement this algorithm[7]. The algorithm works by adding a step before the normal

map generation procedure where the diamond-square algorithm is essentially reversed, the algorithm goes through every data point and checks if their direct ascendant have been generated, if this is not the case the ascendant is generated based upon all its known direct descendants. This continues until every defined cell has had every ascendant generation generated. Then the algorithm runs through as normal. We implemented this algorithm with one change change to it. The algorithm as described worked by adding all initial defined cells to a queue and processing them in order, adding subsequent defined ascendant cells to the back of the queue. This has the consequence that a cell may have had its direct ascendant processed before itself because the descendant that added it to the queue was to far down the generational tree. The result being that some info is lost as some cells cannot contribute to the value of the early generations. As the generation of a cell is completely predicable due to the nature of the diamond square algorithm we decided to to the bottoms up step of the algorithm starting with what would usually be the end generation and continuing to the top.

1	7	5	7	3	7	5	7	1
7	6	7	6	7	6	7	6	7
5	7	4	7	5	7	4	7	5
7	6	7	6	7	6	7	6	7
3	7	5	7	2	7	5	7	3
7	6	7	6	7	6	7	6	7
5	7	4	7	5	7	4	7	5
7	6	7	6	7	6	7	6	7
1	7	5	7	3	7	5	7	1

Figure 5.3: Generation numbers in the midpoint displacement algorithm

HDR Compression Applied to Terrain

In this project we only concern ourselves with the representation of the surface of the earth above water. Even then the height of the surface of the Earth range from 418 meters below sea level at the shore of the Dead Sea to 8848 meters above sea level at the summit of mount Everest[3]. An unmodified version of Minecraft have a maximal vertical range of 256 meters with sea level usually being at 63 meters[2]. This leaves us with an obvious problem when it comes to representing the Earth in Minecraft.

If we want a map consistent with the ordinary Minecraft experience we have 193 (maximum height minus sea level) meters to represent everything from sea level to the highest point on the map. In the extreme example of representing the entire possible range of natural heights on the planet we would need to scale everything down by a factor of 46, mount Everest being 193 meters above sea level and the dead sea 9 meters below. This would obscure smaller details and remove a major point of this project, which is recognizing local landmarks from where you live inside Minecraft. We need to represent both small and large terrain features on the same map. This problem has a close

analogue in photography namely HDR imaging. That is representing high dynamic range image data on low dynamic range display mediums.

The only difference between a grayscale HDR photo and a DEM is what the data represents, light intensity or height. They are both almost universally represented as numbers organised into grids where the numbers have no obvious upper bound, and in the case of DEM no obvious lower bounds either. For both types of data features at multiple scales are important and a linear compression would obscure small scale details. An HDR height map visualized as an image would of course be a prime candidate for an HDR image compression algorithm. The question then becomes the degree the resulting data visualized as a 3d environment looks like the terrain it is supposed to represent. The only way for the terrain to keep its shape is to scale it down linearly, which as mentioned earlier entails other problems.

HDR imaging algorithms usually rely on the assumption that the human vision is much less sensitive to global image intensity ratios compared to local ones[4], so that the algorithms try to keep local ratios as close to the original data as possible on the expense of global ratios. This obviously applies to height data represented as grayscale images until you try to visualize that data in a 3d environment. At this point the analogy breaks down as features that are not physically close may be visually close, for example viewing a mountain behind a much closer hill. Representing relative difference across many different scales necessarily means compacting large features more than smaller ones. This means that terrain features visible in a high dynamic range environment may be obscured in the low dynamic range environments as they are obscured by features less compressed than they are, as shown in figure 5.4.

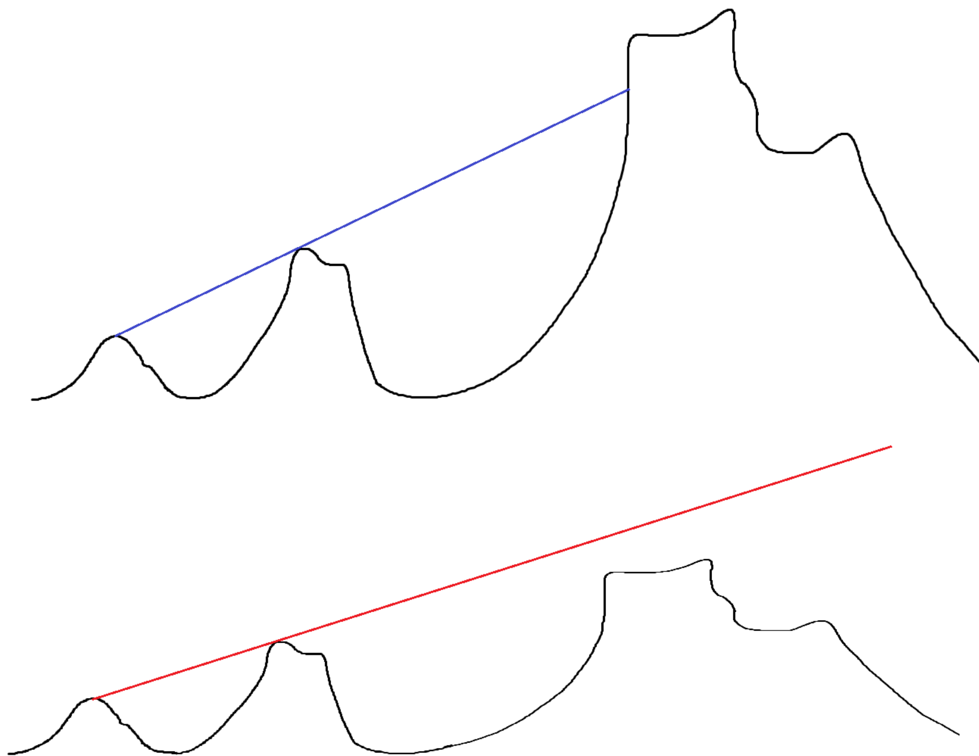

Figure 5.4: Illustration of how non-linear scaling may affect visibility. The blue line represents vision in the real world or a linearly scaled terrain representation. The red line illustrates vision in terrain that has been processed by an HDR imaging algorithm.

It is clear that a non-linear terrain representation becomes to some extent symbolic, but as this is already the case with maps, we must find out how well a symbolic 3D environment works. As the goal is terrain recognition more than an accurate representation, we believe that HDR algorithms will in some cases give us a better result since it can allow us to create maps in a smaller scale than would be possible otherwise and that the distortion in terrain is made up for by the fact that we are representing the area in a scale more familiar to people.

The algorithm we decided to implement was the gradient domain high dynamic range compression[5]. The algorithm works by finding the gradient of the logarithm of the image. The magnitude of the gradient is reduced in such a way that large gradients is reduced more than small ones, this is called attenuating. The attenuation function is constructed based on the original derivatives and smaller scaled down versions. This is so that the derivative of strong edges affect weaker edges nearby so the strong edges won't introduce halo artefacts as they are scaled down much more than the weaker image features around them. The image is finally reconstructed from the attenuated gradient.

The reason for choosing this algorithm in particular is twofold, the focus on avoiding introducing artefacts such as halos around strong edges and the focus on keeping all gradients in the same direction.

In implementing this algorithm[5] there were some problems. In some image processing algorithms, the way you handle edge cases is almost immaterial, you can concentrate solely on how the algorithm performs on the interior. This was certainly not the case in this one. It turns out that this algorithm is extremely sensitive to missing data. For example if we run the image in figure 5.5 through an earlier configuration of the algorithm we end up with the the image in figure 5.6.

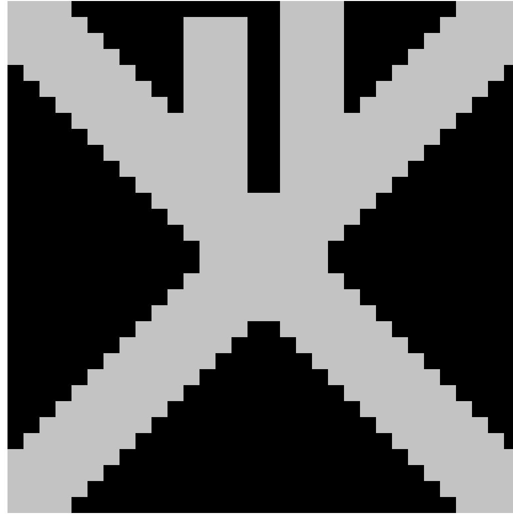

Figure 5.5: The image used in this example, scaled up 800%.

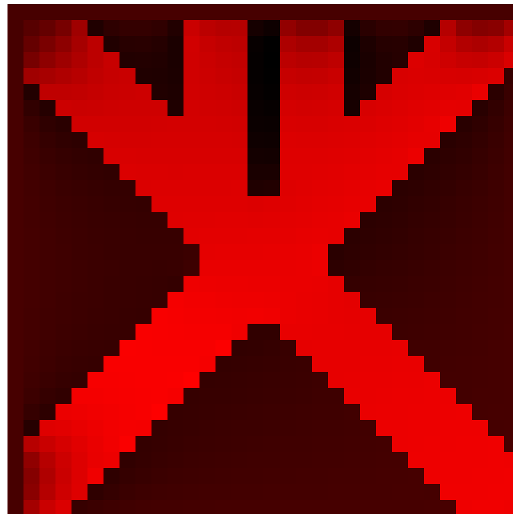

Figure 5.6: The result from the original configuration of the algorithm.

We see that the image has lost the upper row and leftmost column of data and gained some unwanted halo artefacts. This configuration of the algorithm uses a border that is one pixel wide, all the sub algorithms work on the entire area of the image that was part of the original data. In debugging this algorithm it was very useful to export intermediate data as images.

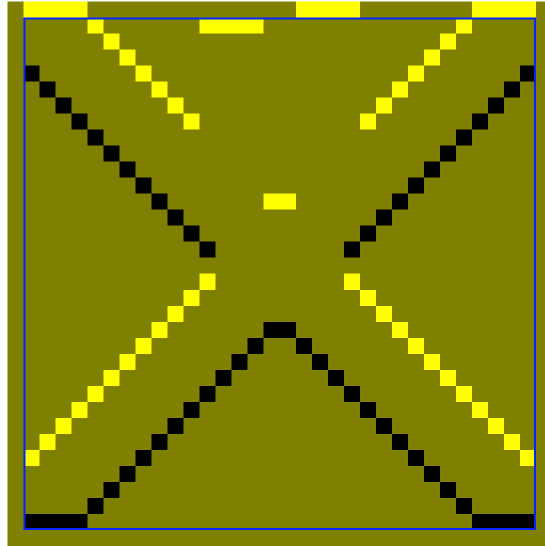

Figure 5.7: The forwards y differentiation of the image.

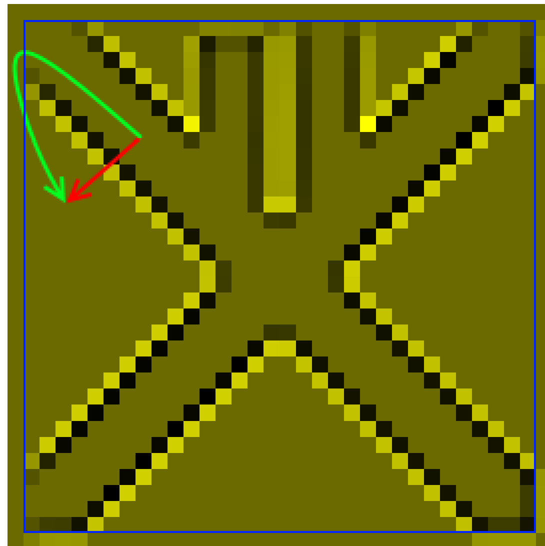

Figure 5.8: The backwards divergence of the attenuated gradient.

On image 5.7 and 5.8 the blue line represents the place where the original image ends, everything outside the blue line represents data used by the algorithms for handling edge cases.

$$divG(p) = \nabla \cdot G(p) = \frac{\delta(\Phi(p)H_x(p))}{\delta x} + \frac{\delta(\Phi(p)H_y(p))}{\delta y} \tag{1}$$

The H represents the log of the image. The attenuation function is represented by the symbol Φ. The p represents a point in the data.

The upper right corner of the image in figure 5.8 represents the divergence of the attenuated gradient explained in formula 1. It is missing data outside the part of the image representing the original data. The problem is caused by the fact the algorithm is

trying to construct a height map where there are completely flat paths and paths leading down a steep slope with the same start and end position, represented as a green and red arrow on figure 5.8. A geometric impossibility in euclidean space. The closest mathematical equivalent is the effect we see, a halo added to the image.

The problem is that the attenuating function was not handling the data outside the original image. It works this way to avoid having to handle edge cases. This means that we are losing the data on the edge when attenuating the gradient. The solution was to increase the size of the image border by two pixels instead of one, so that the there is enough buffer space for the differentiation function to generate all its data and so that the attenuation can work on all the differentiated data and still have enough buffer space to avoid handling edge case.

Since we are using finite difference methods it seems prudent to mention how the various functions are approximated.

(2) $$H_x(x,y) \approx H(x+1,y) - H(x,y)$$

(3) $$H_y(x,y) \approx H(x,y+1) - H(x,y)$$

(4) $$divG(x,y) \approx \Phi(x,y) \times H_x(x,y) - \Phi(x+1,y) \times H_x(x-1,y) + \Phi(x,y) \times H_y(x,y) -$$

This algorithm tries to construct the closest approximation possible from the adjusted gradient, this has the result that areas that are flat in the input data are not entirely flat in the constructed output image. This becomes a problem for areas that this is very noticeable, such as bodies of water. It became necessary to add a step to the algorithm to enforce the flatness of such areas. It works by treating entire areas as a single pixel, any adjustment to any of the individual pixels that make up that area is applied to all of them with a factor inversely proportional to the number of pixels in the area.

In[12] they show that.

(5) $$divG = \nabla^2 I$$

The approximation

(6) $$\nabla^2(x,y)I \approx I(x+1,y) + I(x1,y) + I(x,y+1) + I(x,y1)4I(x,y)$$

gives us

(7) $$I(x,y) \approx (I(x+1,y) + I(x1,y) + I(x,y+1) + I(x,y1) - divG(x,y)$$

The image is really reconstructed through a multigrid method but this simpler but slower single grid version shows us the principle.

(8) $$I_0(x,y) = 0$$

(9) $$I_{n+1}(x,y) = (I_n(x+1,y) + I_n(x-1,y) + I_n(x,y+1) + I_n(x,y-1) - divG($$

After running equation (9) enough times you get a close approximation of the the mathematically closest image which gives $divG$.

Level Set Noise Reduction

It became apparent that we would need a noise reduction algorithm in our project as the DEMs contains some artifacts from the data collection. We choose a noise removal algorithm meant for images on the assumption that image noise would not behave significantly different from the noise found in DEM data. Since we are using the data for a purpose that makes apparent similarity to humans the relevant focus, potential small systematic errors will not be a problem. Had the data been used in a way that made high accuracy more important we would have spent more time considering this matter.

The choice of the Image processing via level set curvature flow[1] in particular had much to do with the fact that this algorithm stabilizes to a final result, meaning that it is much more useful in an automated system compared to an algorithm that would for a better result need human intervention to stop it from removing to much of the actual terrain features. Running this algorithm for any additional iterations after the image has stabilized will not cause any additional changes. This means that configuring the system is much easier as we only have to consider a lower bound on the number of iterations for the algorithm to be effective.

The idea behind this algorithm is to see an image as a series of contours that can be evolved based on the presence of noise until the image stabilizes.

On contours, convex areas have a negative curvature while concave areas have a positive curvature. If we let a contour evolve under its own curvature, along the normal of the contour, concave areas will grow until they become convex and convex areas will shrink until they disappear. Very strongly convex or concave areas indicate noise but allowing the image to evolve like this will only erase it.

If we only allow positive curvature to have an effect we will end up with concave areas growing until the shape is convex. This is known as min flow, min(k, 0). If we only allow negative curvature to have an effect we will end up with convex areas shrinking until the entire shape is convex and then until it disappears completely. This is known as max flow, max(k, 0). In order to selectively remove the unwanted features of the image we need some way to selectively apply either min or max flow. The method suggested was to utilize a switch based on local image properties. The function selects min flow on locally convex points and max flow on locally concave points.

Let $v(\vec{p})$ be the normal speed of the level set curve $\Gamma = \{\vec{p} | \vec{p} \in \mathbb{R}^2, u(\vec{p}, t) = c\}$.

This is can be mathematically proven to be the equivalent of the following[12].

$$u_t + v|\nabla u| = 0 \tag{1}$$

The core of the algorithm is to let contours of the image I evolve along the along the normal direction with the speed F, we then get.

$$I_t = F(\kappa)|\nabla I| \tag{2}$$

I_t is the time differentiated image intensity, $|\nabla I|$ is the magnitude of the image gradient and $F(\kappa)$ is the speed function. This is the equivalent of the image contours moving in the normal direction with speed $F(\kappa)$[1]. Applying finite integration to this gives us the the following.

$$I_{n+1} = I_n + F(\kappa)|\nabla I|\Delta t \tag{3}$$

The variable Δt represents the intensity of each step, we found that a value of 0.02 worked well. Having it higher than that sometimes lead to small images not stabilizing and simply disappearing.

The speed function selects whether to use min or max flow, it it defined as follows.

$$F(\kappa) = \begin{cases} max(\kappa, 0) & \text{if } a(x,y) \text{ ¡ } G(x,y) \\ min(\kappa, 0) & \text{otherwise} \end{cases} \tag{4}$$

The κ symbol refers to the curvature in point $(x, y).a(x, y)$ refers to the average image intensity around point (x, y), $G(x, y)$ refers to the average image intensity along the tangent of the contour in a small radius. The way to find the tangent is simply to take the orthogonal to the direction of the gradient in that point.

The main problem in implementing the level set method applied to noise reduction was related to the speed selection function. The problem was that the function was not sensitive enough so very little would happen to the image, it was difficult to get the function to detect edges between 90 and 270 degrees.

In order to understand the problem it was useful to consider what would constitute a detection. There are four possibilities, the curvature may positive or negative, the iso-intensity tangent may be higher or lower than the local average. A combination of these two gives four possibilities. Looking at the function it is easy to see that two of the combination would constitute a detection. Positive curvature and the iso-intensity tangent being higher than the average leads to $F(\kappa)$ being positive. Negative curvature and the iso-intensity tangent being lower than the average leads to $F(\kappa)$ being negative. The two other combinations gives $F(\kappa)$ equal to zero.

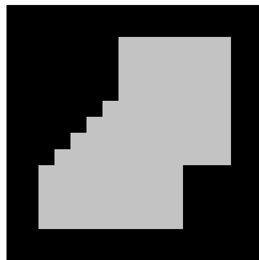

Figure 5.9: Raw data scaled up 800%

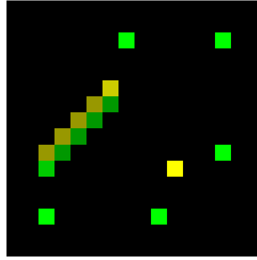

Figure 5.10: Curvature, yellow being positive and green being negative.

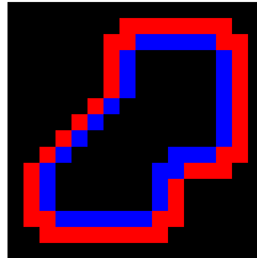

Figure 5.11: Flow selection in an early version, blue meaning min-flow and red meaning max-flow.

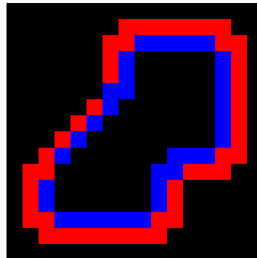

Figure 5.12: Improved flow selector.

In order to understand the behavior of this algorithm we must first understand the properties of the curvature in a point.

The curvature in two dimensions can be expressed as the divergence of the unit normal vector. (http://impact.byu.edu/Image%20Processing%20Seminar/FiniteDifferenceNotes.pdf)

(5)
$$\kappa = \nabla \cdot (\frac{\nabla I}{|\nabla I|})$$

(6)
$$\kappa = (\frac{\delta \hat{\mathbf{i}}}{\delta x} + \frac{\delta \hat{\mathbf{j}}}{\delta y}) \cdot (\frac{I_x \hat{\mathbf{i}} + I_y \hat{\mathbf{j}}}{|\nabla I|}) = \frac{\delta}{\delta x}(\frac{I_x}{|\nabla I|}) + \frac{\delta}{\delta y}(\frac{I_y}{|\nabla I|})$$

Giving us.

(7)
$$\kappa = \frac{I_{xx} \times I_y + I_{yy} \times I_x - 2 \times I_x \times I_y \times I_{xy}}{((I_x^2 + I_y^2)^{\frac{3}{2}}}$$

This reveals that when both the x and y axis curve downwards the curvature will be negative, when both curve upwards the curvature will be positive. If the signs differ the sign of the curvature will be undetermined but the magnitude will be much smaller. Figure 5.13 illustrate that the curvature will be positive in the depressed area in a concave feature, similarly an elevated area in a convex feature will have negative curvature.

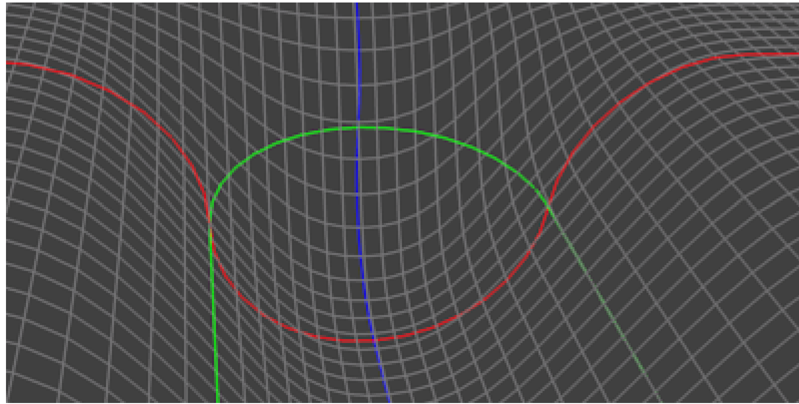

Figure 5.13: Image illustrating curvature in a concave area

It becomes clear that in order to increase sensitivity we need to increase the probability of $a(x, y)$ being less than $G(x, y)$ when on an elevated area and the opposite when not. Considering that decreasing $a(x, y)$ when on an elevated area and increasing it when in a depressed area will accomplish this. Not counting the central pixel when when calculating the average of the local iso-intensity tangent will make the function more sensitive, which solves the problem. This gives us the following result.

Figure 5.14: Noise added to original image data in figure 5.9.

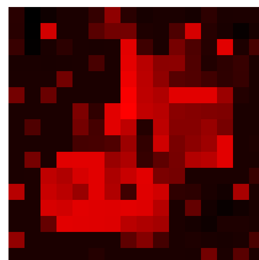

Figure 5.15: Final result using the original function.

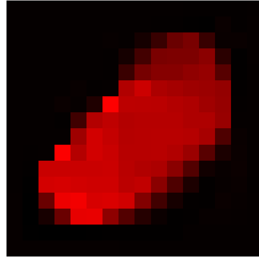

Figure 5.16: Final result using the improved function.

Map Analysis Algorithm

We decided that it would be useful with an algorithm automatically the degree to which different parts of the maps where accessible from each other. The algorithm works by dividing the entire map into convex areas smaller than a certain number of cells we will call c, here defined as areas that every point in the area can be reached from every other without having to take a detour. Note that by this definition moving from on point on to another in a region may involve moving through another region, as long as it can be done in a straight line. As it is possible to jump one block up in Minecraft that is considered the limit for traversable terrain. Every point that borders a region and is accessible from another region is added to a list. Then Dijkstra algorithm is run with every region as a starting point, having a set value as a cut off point for continuing the search called r. The amount of nodes in regions accessible from the starting region is tallied, we will call that value n. When calculating the accessibility A of a certain region we use the following formula.

(1) $$A = \frac{n}{\pi r^2}$$

This assumes that regions have no actual size, but this is accurate enough for our purposes as long as r is significantly larger than $log_2(n)$. We originally planned to implement an algorithm creating paths out from some inaccessible regions if there proved to be too many of them, but using this algorithm on our maps shoved that this would be unnecessary as the terrain generation algorithm generally produced very accessible maps.

Figure 5.17: Image showing accessibility in test image.

Open street map feature parser.

We recognized early that constructing maps exclusively from height data would result in rather uninteresting terrain. Trying to generate terrain features such as lakes and woods based solely on the height data would have been time consuming and error prone. For example, figuring out whether a flat area is a desert or a lake would have been quite difficult.

We realized that we needed a source of such data and decided that the best fit was OpenStreetMap[9]. Our program generates a http request from the location data received in the DEM and stores the response. The OSM data is stored as xml data in an hierarchical fashion. The things we parse from the data are nodes, ways and relations. Nodes containing latitude and longitude are stored with an id number. Ways are objects in OSM that store geometric objects such as polygons and polylines as a list of reference s to nodes. Ways that make up polygons simply store the same id reference first and last. Relations are objects stored as references to ways, this is necessary to make up areas of more complex topology. For example a wood with a hole in it could be stored as two ways, one inside the other. Both the ways and relations may have tags in them signifying that they represent some terrain feature. We parse this data and store it as coordinates, lineCollections and regions. Ways that make up a terrain feature by themselves are turned into a lineCollection and a region referencing that lineCollection. Relations are directly translated into regions. Regions describes the type of terrain it represents. Nodes have their coordinates converted into a map relative coordinate system, but with enough precision to tell where they are inside a map cell and not just which map cell they are in. They are stored as a collection of coordinates.

What we end up with is a list of regions and a list of lineCollections and a collection of coordinates. Each region contain references to its lineCollections. Each lineCollection contain references to its coordinates. For each region we create a two dimensional array. The size of the array is based on the scale of the map and size of the region parsed, the easternmost, northernmost, southernmost and westernmost coordinate is found. The

array is created so that it is large enough to encompass the extreme points. For each line Collection in a region we iterate through all of its coordinates.

First we find the line between the first and the second coordinate and then identify all cells where the line passes through the vertical center line of the cells. We proceed to do this with the second and third coordinates, then between the third and fourth and so on. When the last and second to last coordinate have been processed in this manner we move on to the next lineCollection. When all the lineCollections in a region have been processed we sum up the number of times the center line in each cell has been crossed.

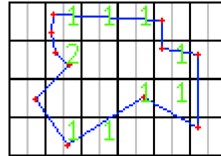

Figure 5.18: An image showing the number of crossings on the array.

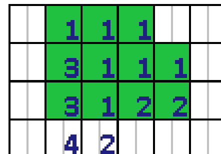

Figure 5.19: The crossing per column summed sequentially with the cells inside the feature marked in green.

We sum up all the cell crossings in each column sequentially from the top to the bottom. Each cell now contains the number of crossings that has happened in it and all of the above cells. A cell with an odd number of crossings in and above it, is inside the terrain feature. If there are an even number of crossings the cell is outside.

The cells inside the region is then transcribed to the main map, detailing the relevant terrain type.

RealMapsWorldType

Language: Java

Java as the language of choice wasn't much of a choice really. As the modding framework Forge/ FML and Minecraft is Java based, the only 2 viable options was Java or Scala. On top of all this, of these options, only Java was viable based on past experience with this language on our team. As a result, it meant that most of the core code was in Java.

Framework: Forge/FML

The choice of Forge/FML was based on that one of our team members had previous experience with using this framework. It is also one of the most used modding framework for Minecraft and has a considerable amount of uses and things one can to to

the game.

Library*: JSON*

Since we decided to use JSON as in our data format and the most practical library for this was json.orgs own implementation.

Build system: Forge Gradle (custom Gradle for Forge specifically)

This choice is the result of using Forge/FML. As the framework has its customized version of Gradle to set up development environment and building working mod files, means thet there is no piratical way around it. Not that it is a bad thing as it means all the things that are required to mod for Minecraft is taken care of.

RealMapsWorldType is a Minecraft mod. The purpose it serves is to use data from RealMapsGenerator to generate worlds in Minecraft to those specifications. It's mainly intended as a server side only mod but will work client side as well.

The Mod is made to be used with Forge, a highly popular modding framework for Minecraft. By it being a mod, it easily integrates itself into Minecraft's already existing code base at runtime. This gives us access the already existing default Overworld terrain generator. What we did is creating another WorldType that extended the default one. By doing so, we supply our own generator that extended the default generator, making it so that we could manipulate anything in the generator process.

When our generator is asked for a chunk, the mod checks if the chunk is inside the area defined by the RealMapsGenerator. If that is the case, it replaces all the blocks of the first stage with another set of blocks that mirror the data given by RealMapsGenerator for that area, including water height. Something similar happens when it comes to the biome step in the generator chunk generating process.

The mod is configured via a .cfg file in the config folder, and the "server.properties".

The mod also has client side only features (mostly for debugging) like the ability to select a dataset interactively via the graphical user interface when configuring the generator.

Configuration

Configurations for RealMapsWorldType is done in 2 files. Additionally, there is a client only interactive way to configure these settings as well. The client only thing relates to the "Customize" options when selecting world type as it allows the player to select what dataset to use from a list of those parsed from the data folder.

The general settings file in "<Minecraft installation>/config/RealMaps.cfg"

```
# Configuration file

general {
    # The folder to search after valid data sets in
    S:data_folder=terraindata

    # Name of the data set that should be selected as default
    S:default_data_set=default

    # Force all areas in data set to be generated
    B:force_generate=false

    # Shut down Minecraft after force generating the areas
    B:force_generate_quit=false
}
```

All of these options should be self explanatory with their comments. Note that the "force_generate" and "force_generate_quit" only works on dedicated servers and not when running the integrated one as a client.

The next set of configuration options is only for dedicated servers as they are in the "server.properties" file. The options that will affect the generator are "level-type" and "generator-settings". "level-type" is the field that determines what generator should be used when creating a new world save. If the field "generator-settings" has a value beyond being empty, then the value it has will be used as a path to a dataset to load and use, otherwise the "default_data_set" value will be used if possible.

There is 1 more thing to take note of and that is the file "level.dat" in a saves structure. If that save is using this world type the the field "generator-settings" will be used in the same way as said above.

Generator

There actually exists 2 different version of the WorldType, "RealMaps-Flat" and "RealMaps-Default". The difference is that the flat variant sets the ground level to 4 when that point is outside the dataset selected for that world. The default variant uses the default generator with its settings when that column of blocks is outside the dataset.

Initialization

The first thing of note happens when the mod initializes. The mod will check its settings for what folder is its default data folder. It will then scan that folder for all its valid dataset and store a reference to them.

When the server component of the game starts up, given that the world type is set correctly, the mod will load the given dataset specified in the "generator-settings" as stated earlier. This field also exists in already created worlds so that when they are loaded again the correct dataset is loaded. Note that this only works as long as that

dataset it used stays at that path. If it is a fresh world then the field in the "level.dat" file is set to the path of the dataset selected to be used.

Generating chunks

All this presumes that the world this server has loaded is using RealMaspWorldType as it world type. When the server tries to load a chunk it checks to see if it exists, if not the server will request that chunk from the chunk generator. And in our case, is replaced with RealMaps version of it thanks to the world type in use. When receiving requests for chunks the first thing that is done is to check if it is within bounds of the defined area of the selected dataset. If it is outside then the chunk generated is based on the default behavior, and that is based on the version in use. If on the other hand then the chunk is generated in accordance to the data for that area specified by the selected dataset.

In the first part of generating a chunk, each column of blocks are made to specifications. First the column are filled up with stone blocks to height as specified by the dataset for that point and after that, all air blocks up to the specified water level are replaced by water blocks. In a later step where the distribution of biomes are supposed to be laid out, the biome distribution for that chunk is replaced by data from the selected dataset that it has for that area. All other things are handled by the default generator for Minecraft, resulting in rather natural looking world with caves, forests and villages.

Other uses

The mod includes a chat command for translating the current player's positions into latitude and longitude and then hand it to the player in the form of a Google Maps link. The result is only useful if used when using a save where RealMapsWorldType is in effect.

When it comes to the datasets one could artificially create them as long as it fits the format definitions. This means that other people can use it to create almost custom worlds that takes a fraction storage space compared ordinary custom save files.

Website

Language: Python, bash

The main reason behind using Python is simply that Django is python based.

Framework: Django

After looking at many different solutions, the one that had support for what we needed and looked flexible enough without needing to learn an entirely new language was Django. It has a stable and hassle free database support, management and integration. It also has very nice templating and page rendering systems.

Library: Celery, JSON

Celery is a task queuing and distribution library made for real-time operation and scheduling tasks as requested. Tasks, or sometime referred to as execution units, are executed on any of the currently available worker servers set up.

Programs: PostgreSQL, Apache.

PostgreSQL was used as the database solution, mostly because its was nicer to work with. Apache was used as it was the default and Django works with it by using mod_wsgi.

This project as stated earlier, included the thoughts of generating these worlds as a service. One of the challenges with this based on the fact that a Minecraft server is needed as part of generating the maps. This meant either handling multiple Minecraft instances running in parallel or just 1 and some locking mechanism. As this site would only be a prof of concept, the simpler solution with managing 1 server was chosen. The other option would work but more system resources and development time was needed for that.

Generating worlds

The process for a world to be generated starts with a job request via the forms. In this form the user can specify the geographical data to use, what settings to transform the data with, name and description for the job. After this is recorded in the database, a celery task is set up and executed on one of the worker threads. First thing the task does is to create the folders used to store temporary and permanent results in. Next is to run the generator to create a RealMaps dataset for the mod to use with, where all the options set earlier at imputed as launch parameters for the generator to use. When the dataset is generated, the Minecraft server is configured properly and then started. The server will work until the entire map is generated and saved to disk. At this point the world file is zipped up and moved to storage, the job marked as done and results recorded, and a download link is generated.

User Testing

In order to test recognizability of the maps we have designed a set of experiments. The point of the experiments is check whether a point on a conventional map may be found inside Minecraft and vice versa. Given that we know the scale of the map and the latitude and longitude of the map centre, it is relatively easy to convert Minecraft map coordinates to real world coordinates and vice versa. The map we decided to use was Google maps, it being the easiest alternative. Also, during testing we had a logger running to track various things about the subjects in game. These was in our case what

direction they where looking and their position. Our goal for this was to see how close they guessed the location and how they got to that location. This should hopefully give insight in how much users recognizes their surroundings in Minecraft when compared to a real place.

Test 1

The test subject is given a map with a location marked, the subject is asked to find the corresponding points in a Minecraft map representing the same area.

Test 2

The test subject is given a Minecraft map with a location marked, the subject is asked to find the corresponding points in a map representing the same area.

6. Result

User Tests

The images below shows illustrates the path some of our test subjects took. The red dot is the start location and the blue dot is the end location.

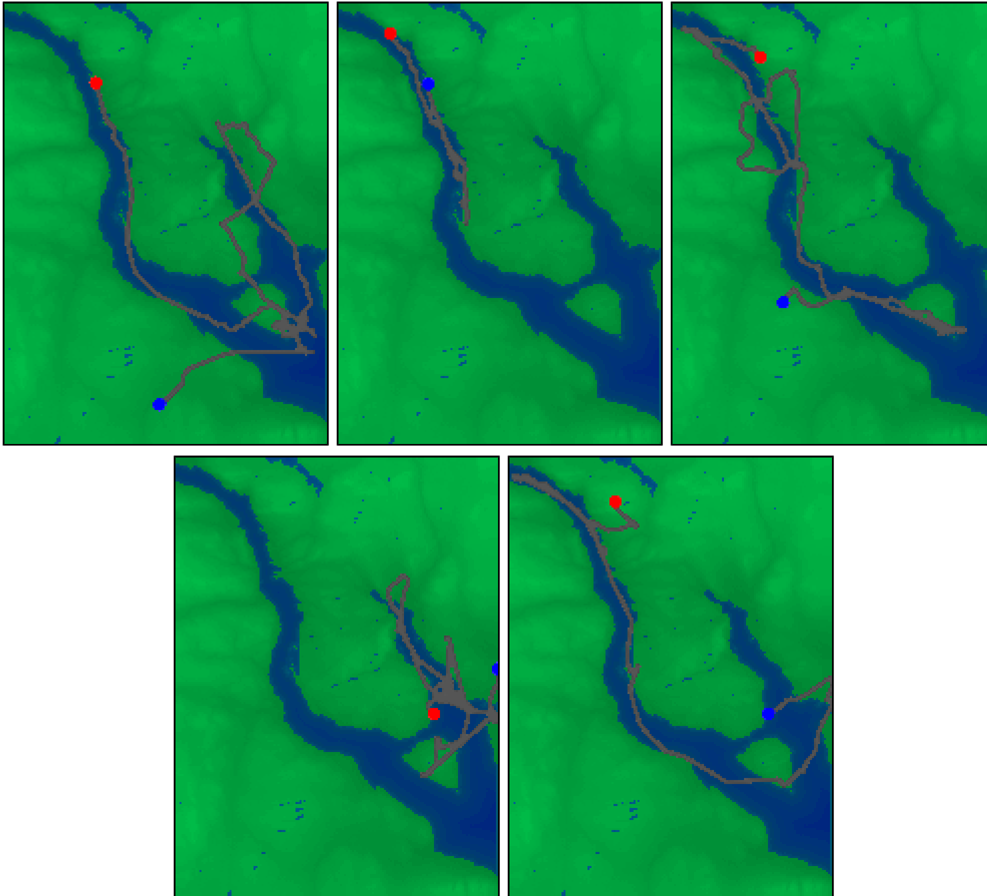

Figure 6.1 to 6.5: Assorted collection of paths taken by test subjects.

We did two main rounds of tests, one with children around the age of ten. And one with people in their early teens.

Generally we discovered that people in their early teens and children have some difficulty navigating the maps. The fact that the around four year age difference between the groups of test subjects made such a little difference was surprising. We did a few informal tests with college students, they much less trouble navigating the same maps. Most people in all age groups had roughly the same approach, try to identify a terrain feature in water and navigate based on that. Even though the rate of success varied with the age groups generally the strategy was the same, find a very obvious feature on the map and navigate based on it.

2	L20						572.448 309.495	1518.500 986.500		
1	G2010					60.864184, 10.695748				
3	L20								1029.300 801.700	1491.403 978.644
4	L20	2	4	4	3		511.594 1381.726			
5	G2010					60.837069, 10.750397				
7	G2010	1	3	2	3		519353			
6	L20			3					---	
8	G2010	4	4	1	5			10951286		
9	L20	3	5	3	4			715.511 1854.992		
t2-1	L20	4	4	3	4		1352.74 2006.771	778.300 1533.320		
t2-2	G2040	5	4	2	5					1706921
t2-3	L20	4	3	3	5		769.692 1476.320	115.769 1739.556		
4	G2040	3	5	1	5				1125964	
t2-5	L20	4	5	3	4		836.123 1542.881	968.231 1746.870		
t2-6	G2040	4	4	4	3				439.260 1245.512	outside

1. Tester
2. Map Type
3. The map looks like Gjøvik
4. The map looks like Minecraft
5. It's easy to navigate the map
6. The map looks nice
7. Find the Latitude Longitude of the Beacon in minecraft
8. The Minecraft x z Coordinate of this point. https://www.google.no/maps/dir/60.797420,10.938327//@60.797420,10.938327,10z

9. The Minecraft x z Coordinate of this point. https://www.google.no/maps/dir/60.789137,10.682920//@60.789137,10.682920,10z
10. The Minecraft x z Coordinate of this point. https://www.google.no/maps/dir/60.720427,10.805908//@60.720427,10.805908,10z
11. The Minecraft x z Coordinate of this point. https://www.google.no/maps/dir/60.834592,10.618659//@60.834592,10.618659,10z
12. The Minecraft x z Coordinate of this point. https://www.google.no/maps/dir/60.847043,11.150257//@60.847043,11.150257,10z

7. Conclusion

Future Work

When we started this we had many ideas for possible features that we did not have time to implement, the one core feature we did not manage to implement was automatic downloading of the geographic height data. The API for scripted downloading of the data was non-existent. Also the program as is supports only one data- format ARC-ASCII, extending support for other formats would have been high on our list of priorities. For the worlds generated, we had plans for roads, markers and borders to populate the world with. Another direction worth exploring is generating specifically for popular mod packs, like the "DireWolf20" pack by "Feed The Beast". Doing things like generating structures and using biomes specific to that collection of mods. Another thing that needs further work is the website itself. Because the site is only a prof of concept it lacks in visuals, design, interactivity, usability and reliability.

User Testing

When conducting the tests we mainly learned that conducting QA tests properly and consistently is hard and requires quite a lot of planning.

It is difficult to draw more conclusions than that based on our rather small sample, but to the degree we can conclude anything we can conclude that norwegian children and early teens are generally not very good at orienting on our maps. Observing the tests it became clear that most people relied on obvious water features to navigate, most of the subjects managed to find and identify the island Helgeøya but often got disoriented when moving to far away from it. The testers where obviously hampered by the relatively short view distances possible in Minecraft in combination with the relatively the relatively small downscaling of the maps we where using. We did not have time to tests maps that did not have bodies of water in them but can only assume that they would have been even more difficult to navigate. We can in any case conclude that additional landmarks such as roads and the location of cities would have been helpful.

An interesting possible follow up test could be to give the testers some training in navigating and check if their performance improved.

General Conclusion.

Our project generates maps generally consistent with Minecraft gameplay and aesthetics. The terrain is recognizably similar to the terrain it is supposed to represent. While some people may have trouble navigating the maps initially, we are confident that adding additional landmarks or other features representing the real world like major roads will solve that issue. When it comes to offer this as a service, the website works but our

current limitation on what geographical data that users can select from makes it rather useless. This will fix itself when more options to adjust for generating maps like structures in the world, different way to compile data and more diverse collection of geographical data to select areas from.

Figure 7.1: Google Maps image of Helgøya

Figure 7.2: Helgøya in scale 1:25 gradient domain 1:60

Figure 7.3: Helgøya in scale 1:100 Linear

8. References

1. Malladi, R., & Sethlan, J. A. (1995). Image processing via level set curvature flow. (http://www.pnas.org/content/92/15/7046.short)
2. Minecraft Wiki. http://minecraft.gamepedia.com/Altitude. [Online; accessed 4-May-2015].
3. Wikipedia. http://en.wikipedia.org/wiki/Extreme_points_of_Earth#Highest_point. [Online; accessed 4-May-2015].
4. Dicarlio, J. M., & Wandell, B. A. (2000). Rendering high dynamic range images. Proc. SPIE 3965, Sensors and Camera Systems for Scientific, Industrial, and Digital Photography Applications.
5. Fattal, R., Lischinski, D., & Werman, M. (2002). Gradient Domain High Dynamic Range Compression.
6. Seetzen, H., Heidrich W., Stuerzlinger W., Ward G., Whitehead W.,Trentacoste M., Ghosh A., & Vorozcovs A. (2004). High dynamic range display systems.
7. Belhadj, F. (2007, October). Terrain modeling: a constrained fractal model. InProceedings of the 5th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa (pp. 197-204). ACM.
8. Changelog for 1.7.2, Changes -> GamePlay -> World boundary http://minecraft.gamepedia.com/1.7.2#Gameplay [Online; accessed 08.05.2015]
9. http://en.wikipedia.org/wiki/Esri_grid [Online; accessed 09.05.2015]
10. http://diyhpl.us/~bryan/papers2/frey/levelsets/Chopp%20D.L.,%20Computing%20minimal%20surfaces.pdf
11. Osher, S., & Sethian, J. A. (1988). Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations. Journal of computational physics, 79(1), 12-49.
12. Sethian, J. A. (1985). Curvature and the evolution of fronts. Communications in Mathematical Physics, 101(4), 487-499.
13. Minecraft. Wiki. Minecraft Biomes http://minecraft.gamepedia.com/Biomes [Online; accessed 11.05.2015]
14. Wikipedia. Equirectangular Projection http://en.wikipedia.org/wiki/Equirectangular_projection [Online; accessed 11.05.2015]
15. Chunkmapper homepage https://secure.chunkmapper.com/index.html [Online; accessed 13.05.2015]
16. Level Set, Wikipedia http://en.wikipedia.org/wiki/Level_set [Online; accessed 15.05.2015]

9. Appendix

Original design notes, Ben Sawyer

Edit Document

RealMaps for Minecraft

Goal : Make it possible to create **HIGHLY playable** maps from real geographies using publicly accessible geographic data enabling new types of play and learning inside Minecraft.

Initial Notes & Thoughts

Looking at GeoBoxers and thinking about what we discussed it does seem like what we should be focusing on is a level of ease, speed, and automization with playability vs.a level of accuracy. Accuracy of some level is needed but what we really want is to make it easy for someone to get something reasonably accurate and fun to play and experiment with.

What we want this project to do is the following:

1. It's geared toward kids, and the public-at-large. The maps created need to be meaningful and easy to produce, and navigate once in game.

2. The maps need to be playable, that means easy to build on, easy to recognize the area if you're familiar with it, and have navigable traits such as not having forests that are too hard to navigate, featuring key bodies of water, rivers, lakes, ponds, and having some ability to place down key areas of interest, not buildings, but ideally major highways, place name locations, city centers, etc.

3. They need to allow mashups. So once you render the data, it'd be great to allow the surface layer to be just sand, or just dirt, or all ice, etc. We should be trying to build topology spines, that then can be rendered out in different ways that people find interesting, certainly one skin could be realistic terrain types, but we shouldn't think our end goal is only this realistic portrayal.

Toward those ends here are some initial thoughts:

1. We do not want to render buildings. The results are not necessarily that interesting in some respects because the details will suck. This was clear in Chunkmapper as well.

2. We need to really avoid vegetation that isn't done as Minecraft does it now. We want to use Minecraft's trees and others and not interpret our own. Chunkmapper's vegetation approach is terrible.

2. We do want to render major roads as roads provide great geographic markers for young people and people in general as to where a place is. How we are able to layer road information and cull it to just major roads so cities aren't over run with streets is a critical need.

3. We do want to try and provide place data, ideally it'd be signs that are set down at specific points with names on them, and another idea is "circles" denoting say a city limits or just a city center. Again culling for major vs. minor or medium level names will be key.

4. We want to preserve the entire underground aspect of Minecraft as it is which is what people love about it when they play if they're not playing some specific MOD. The best way to do this would be that that we are able to merge a Minecraft generated below-sea level generated world with our above sea level generated world.

The way to do that is to use Minecraft's considerable map customization system to generate 1 or more maps that has sea level placed at say 30, 40, 50, 60, 70, 80 (a couple that >64) and then see them generated out to some X * Z level and then to use those as that surfaces upon which to paint the above surface generated worlds. This creates the best of both worlds, Minecraft's well honed < sea level creations with our above world real-world facsimiles. If someone doesn't want caverns and dungeons, etc. they can simply marry it to a sea level of 0 or a higher sea level but with a simple custom map of below surface stone, sand, etc.

5. We want to render topology most of all, and within that we need to think about how we handle height maps. I feel like what we want to do is as let the user set the height map, and maybe also select the exaggeration of it all as a result. Let it be relative accuracy but with the ability to add stress to higher points or lower points. So if normally a hill outside of a town is 10 bricks high a stress on anything > 5 might be 2X making the hill 20 and the mountains further offer 120. One idea especially would be to have midrange heights push to be closer to higher items. So if the Alps are 150 make a midrange not 75 but 100. While anything below stays more accurate. The idea being we can allow people to not have only the tallest elevations be tall. Minecraft is more fun when there are more elevations.

Minecraft itself already experiments with exaggerated topologies too so it's not a stretch to realize it has value.

6. I think we should first figure out how to just represent a 3D topology in one bricktype first before figuring out how to render topologies with more accurate bricktype interpretations. However, soon thereafter we're going to want to figure this out better and create some specific interpretations how we render or allow people to define things like roads, forests, incidental trees, etc.

7. Boundaries are another issue. It would be good if we could allow people to select the ability to erect a wall that places itself down using county, city limit, state/province, country markers. That way they could truly denote the beginning and end of areas vs. see it bleed over and not have any sense where the boundary is when play. ESRI's "open" shapefile format might work well here to make it easy to create boundaries of countries, counties, states/provinces. I might have access to a key person there but have to dig it up if we go that route.

8. We want the ability to really have maps that zoom in close or not. Chunkmapper gets close but I think needs to be about 50% closer in scale. I really think we want unlimited zoom in some respect just not unlimited size. E.g. if we choose to limit our Minecraft world sizes to 10,000x10,000 then if you choose to do a 10km to 10km stretch you're down to a 1 brick = 1 meter interpretation of the world. We'll need to figure out the realistic map size we can support, I can run some tests on that in terms of map sizes, etc. let alone speed to calculate.

9. It seems to me what we would do is take some sort of NASA topology data and then render that, and not put trees down, then we'd layer over that major roads as garnered from Open Street Map, or some other valid source. Maybe there is a simpler method using .KML files from Google Earth exports, or some sort of easier to find global elevation data that is just precise enough. The most detailed topographic data is NASA's ASTER data,

http://gis.stackexchange.com/questions/17989/how-to-download-the-entire-aster-gdemv2-dataset

It seems to me what we'd want to do is first just get a nice clear chunk of this data and then figure out how to do that, then if we made that work we'd figure out how to download the entire dataset, or get an HD ordered with it.

At the same time I feel like what we really want is perhaps a version of this data that we've re-processed into something simpler and quicker to generate maps from but which benefits from the rich terrain data it has. Perhaps someone has already done that but not sure where to find it, ESRI might have something, or older topographic data sets might be good enough. My sense is though things that do basic elevations will be hard to model from in terms of making them fun for Minecraft.

10. I think our best approach again, will be to keep refining this paper plan, and doing some exploratory, until we find the right fit of data, and output relevant to our goals.

11. In my ideal world, there is a web site, you go to the Web site, and you can select a boundary square, or circle, or esri shape file and then a set of options like sea level, height exaggeration, surface features, and a few other options, and what it does is a cloud service renders the map and makes it available for download (ideally for a price to keep it sustainable) and then I take that file and I load it into Minecraft and I have fun.

So this is a start, perhaps we toss this into a google doc, and further it from there.

Initial thoughts and ideas

Initial thoughts

- Minecraft Server with Forge
  - Utilize vanilla underground generating
  - Custom terrain mod
  - NBT-Lib
  - heightmap
  - pipeline / node tree for easy process manipulation
- Web page
  - User input -> map

Prototype ideas

- png heightmap -> dirtmap

After first prototype

- Welding map to standard minecraft map. 2d splines? Multivariate interpolation.
- Map processing.
  - Landmarks visible.
  - Regions: linear planes ax+by=c.
  - Dijkstra’s?
  - Fill regions. Keep them convex regions. Quick hull.
- Map playability score system.
  - Amount of features, distribution, landmarks, similarity to real world, accessibility.
- Quantization of terrain data.
- Terrain spatial compression. Cartogram algorithm.
  - http://ivi.sagepub.com/content/13/1/42.short

Meeting notes bundle

12.01.2015

Date

12 January 2015

Attendees

- Tellef Møllerup Åmdal
- Sindre Helleborg
- Simon McCallum

Things to look into

- Data mashup
- Quantization of data
- Legal aspects
- Abstraction of data
- Hosting costs / set up own server?
- Target: standard Minecraft server
- Read around distance compression
- jpeg reference group: visual acceptance
- Map representation

28.01.2015

Date

28 January 2015

Attendees

- Tellef Møllerup Åmdal
- Sindre Helleborg
- Simon McCallum

Notes

IP ownership
  51 / 49 Ben, Us
  50 / 40 / 10 Ben, Us, Simon
  70 / 10 / 10 / 10 Ben, Sindre, Tellef, Simon

30 hours a week
Contract deadline extended to at least to Friday

% of profit, sum up to a limit

17.04.2015

Date

17 April 2015

Attendees

- Tellef Møllerup Åmdal
- Sindre Helleborg
- Simon McCallum

Agenda

- Confluense setup for bachelor thesis
  - Link to page or PDF of site?
- Progress report
- Practical bachelor and what it means
  - What we write about
  - Science
- Delivery of source code
  - Webpage?
  - Link to repository or tag zip?

Notes

- Biomes implemented
  - Temperated rainforest
- Thesis notes
  - HDR
  - Meeting notes
- Remaining
  - Thesis
  - Automatic geodata download
- For grade
  - Write about automation process
  - Methodes
  - Getting access to data.
- To next week
  - Show of progress
  - Feedback from Minecraft players
  - Make relevant questions.
- World biomes

        - Png map
        - Key values
        - KML, define regions. Geographical markup.
        - Rasterise regions
    - Thesis
        - Results of questions
        - Abstraction and play
        - Elaboration around the original design documented.
        - Interaction
        - Background
            - Mc
            - Scale
            - Users
            - Market
                - Education
        - Objectives
        - Methodoliges
            - How things are applied
        - Result
        - Conclusion
            - What we achieved
        - Think in categories, not temporal.

22.04.2015

Date

22 April 2015

Attendees

- Tellef Møllerup Åmdal
- Sindre Helleborg
- Simon McCallum

Agenda

- Details about thesis writing
- Details about user tests
    - Time
    - Sample size
    - Tests
- Terrain visualizer
    - Unreal

  - Gfx exam based

Notes

- Progress
  - Ideas about tests
    - Google maps to mc world
    - And reverse
  - Trial run for tests with other students
- KML
  - Open street maps
  - Convert mc world tracking data to kml
    - Visualize in google earth
  - Minecraft
- http://www.dwtkns.com/srtm
- http://www.geodata.policysupport.org
- Mc to klm converter
  - Small external utility
- Minecraft mod to convert selected area to klm building
- Thesis
  - Export to HTML
  - HTML to LaTex
  - Look and feel for pdf export
  - Intro
    - Background
      - Hdr
      - Other work
      - Similar systemsystem
      - Jpeg
        - Not absolute accuracy
        - Relative accuracy
        - Symbolic representation
    - Related work
    - Word explanation
  - Methodology (what we aimed for)
    -
      - What we do different
      - Structure by area of dvelopment
  - Implementation (what we did)
  - Result
    - Images
    - Error metrics?
      - Local vs global differences
      - Applied hdr

        - Debug data
            - Iteration diff
- Conclusion
    - Future work