

# Visualisering af funktioner af to variable i Minecraft

Philip Peder Hansen

December 20, 2013

# Contents

<b>1</b>	<b>Summary</b>	<b>3</b>
<b>2</b>	<b>Indledning</b>	<b>4</b>
<b>3</b>	<b>Funktioner af to variable</b>	<b>5</b>
3.1	Afbildning af funktioner af to variable . . . . .	5
<b>4</b>	<b>Minecraft</b>	<b>6</b>
4.1	Generering af verdner . . . . .	6
4.2	Planlgning . . . . .	8
4.2.1	Forge . . . . .	8
4.2.2	Ligning . . . . .	8
4.3	Implementering . . . . .	9
4.3.1	Iteration 1 . . . . .	9
4.3.2	Iteration 2 . . . . .	10
4.3.3	Iteration 3 . . . . .	12
4.3.4	Iteration 4 . . . . .	12
4.3.5	Iteration 5 . . . . .	13
4.3.6	Iteration 6 . . . . .	13
4.3.7	Iteration 7 . . . . .	13
4.4	Vurdering . . . . .	14
<b>5</b>	<b>Perspektivering</b>	<b>16</b>
<b>6</b>	<b>Kilder</b>	<b>17</b>
<b>A</b>	<b>plot.py</b>	<b>18</b>
<b>B</b>	<b>plot1.png</b>	<b>19</b>
<b>C</b>	<b>screenshot1.png</b>	<b>20</b>

D	screenshot2.png	21
E	screenshot3.png	22
F	screenshot4.png	23
G	screenshot5.png	24
H	screenshot6.png	25
I	screenshot7.png	26
J	screenshot8.png	27
K	screenshot9.png	28
L	mcmod.info	29
M	SRPMod.java	30
N	SRPWorldGenerator.java	31

# Kapittel 1

## Summary

For my SRP I've worked with generation of world in *Minecraft* based on functions with two parameters. I've described what functions with two parameters are, how they can be illustrated, and how they're connected to three dimensional graphs.

I've generated an illustration of the function

$$y = \cos(x^2) + \sin(z^2)$$

with *python* and *matplotlib*.

Furthermore I've implemented the same function into *Minecraft's* world generation code, and documented my work to do so.

I've come to the conclusion that functions with two parameters and a good basis for generating worlds, but they're not optimal. For a good world generation a function with at least three parameters is needed, allowing for a more randomized world generation, rather than a fixed pattern.

# Kapittel 2

## Indledning

I min SRP har jeg valt at arbejde med *Minecraft*, og generering a verdner deri ved hjælp af funktioner med to parametre.

Jeg har skrevet om hvad funktioner med to parametre er, hvordan de kan visualiseres, og enda lavet et visualiserings eksempel.

Jeg har eksperimenteret med at generere verdner i spillet ved hjælp af funktionen

$$y = \cos(x^2) + \sin(z^2)$$

Og dokumenteret min fremgang for at nå der til.

Jeg er kommet frem til den konklusion at en nogenlunde verden sagtens kan genereres i *Minecraft* på baggrund af denne funktion, men at en funktion med mindst tre parametre er nødvendig for at lave en ordenlig verden.

# Kapittel 3

## Funktioner af to variable

Funktioner af to variable, er som navnet hentyder, funktioner der tager to variabler, eller input, i stedet for det som vi bruger mest normalt, menlig funktioner af en variable.

Et eksempel på en funktion af en variable kunne for eksemepel være

$$y = x * 2$$

En sådan funktion tager en variable, altså en ting der kan ændre sig  $x$ , og giver et resultat  $y$ .

En funktion af to variable kunne derimod se ud som

$$z = x + y$$

Her findes der en  $z$  værdi for hver to input værdier.

### 3.1 Afbildning af funktioner af to variable

Funktioner af to variable kan afbilledes på flere forskellige måder, for alle er dog ens at der findes en parameter, der afhænger af to andre.

Den simpleste måde at afbilde en funktion af to variable er gennem et tre dimmensionelt koordinatsystem, altsået plan af  $x$ ,  $y$  værdier, og en tredje værdi,  $z$ , der er produktet af de to andre.

Der findes dog andre måder at illustrere funktioner af to parametre. For eksempel kan man farve et punk basseret på punktetes værdi.

Herunder ses en funktion af to parametre

$$y = \cos(x^2) + \sin(z^2)$$

Denne funktion kan tegnes som en værdi på en vertical akse, baseret på et punkt i det horizontale plan.

Se Appendix B for funktionen tegnet med *matplotlib* i *python*.

# Kapittel 4

## Minecraft

*Minecraft* er et videospil lavel af *Markus "Notch" Persson*. Spillet er en blanding mellem et eventyr overlevelses spil, og et kreativt spil der går ud på at bygge verdener.

Spilleren har mulighed for at ødelægge blokke, samle dem op, og placere dem tilbage i verdenen. Man kan møde monstre der slår på en, skyder ildkugler og enda eksplodere, all med formål at gøre det svært at overleve.

I sin kamp mod det onde kan spilleren bygge et sikkert hus, grave efter metaller og smelte disse til jern, guld og diamant brynjer og svære.

Spillet har også et element der hedder *redstone*, der til en vis grad fungere som ledninger. *Redstone* kan bruges til at lave kredsløb der automatisere skydning af pile, hældre lava ud over ens fjænder og er enda brugt til at lave *minigames* i *Minecraft*.

Indledene bliver spilleren placeret i en stor verden, der består af  $1\ m^3$  blokke. Disse blokke har forskellige fysiske egenskaber, og bruges i forskellige sammenhænge.

Derudover genereres blokkene baseret på nogle algoritmer, der resulterer i et mønster der til nogen grad minder om den virkelige verden. Træer genereres i nærheden af hinanden, i skove, og ikke midt ude i havene. Ørkner og tundra findes generelt ikke umidelbart i nærheden af hinanden, og floder løber ofte gennem regnskove.

### 4.1 Generering af verdner

Algoritmerne der bruges til at generere verdener i *Minecraft* er en del mere komplicerede end en det simple eksemepel på den funktion af to parametre vi har kigget på. Ud over  $X$  og  $Z$  koordinater bruger *minecraft* også noget som kaldes et seed til at generere verdener, de genereres altså ud fra en mere



Figur 4.1: En *Minecraft* verden

kompliceret funktion af tre parametre.

Et *seed* er en tekst streng som brugeren kan give *Minecraft* når en ny verden genereres, hvis brugeren ikke giver denne streng bliver den automatisk genereret før verdenen laves. Funktionen af et seed er at selv med den samme verden genererings kode, kan vidt forskellige verdener laves.

Forstil dig for eksempel dette scenario, vi bruger den følgene formel til at generere en kurve

$$y = \sin(x)$$

Denne formel vil altid give en sinus kurve, som vi også ville forvente det, det passer perfect i matematik, men ikke så meget hvis vi prøver på at lave interesaante mønstre.

Forstil dig nu hvis vi ændrede ligningen til at vre

$$y = \sin(seed * x)$$

Så længe *seed* ikke er lig nul, vil denne formel stadig give en sinus kurve, men afhængligt at hvad vi sætter *seed* til at være, vil perioden for vores kurve være forskellig.



Figur 4.2: *Minecraft* verden genererings skærm

På samme måde bruger *Minecraft* dette *seed* i generations koden til at skabe verdener der er unikke, på trods af at de alle er genererede fra den samme kode.

Ud fra dette seed og generations algorytmerne, bestemmer *Minecraft* hvilken blok der skal placeres på hvert punkt i verdenen.

## 4.2 Planlgning

For at implementere vores egen kode i *Minecraft*, og generere en verden baseret ud fra den følgene funktion, er der nogle overvejelser vi først må lave om hvordan dette skal forgå.

Den første, og måske vigtigste, overvejelse jeg har gjort min i forhold til at implementere denne formel i *Minecraft* er hvordan koden skal indkorporeres i spillet.

Da spillet ikke distribueres som kildekode, er det altså ikke muligt bare at skrive koden ind i den orginale kode, og kører spillet. Heldigvis findes der en uofficiel API til *Minecraft*, der gør det muligt at skrive noget kode som en separat file, som bliver kørt når *Minecraft* gør det, denne API kaldes *Minecraft Forge*.

### 4.2.1 Forge

*Minecraft Forge* eller bare *Forge* fugere på den måde at programmet som i sig selv ændre på noget af kilde koden til *Minecraft*, de ændringer åbner op for at andre filer, kendt som *mods*, kan ændre påværdier i spillet, uden direkte at skulle ændre på koden. Dette er især vigtigt da to mods der ville prøve at overskrive to forskellige ting i den samme fil, ellers ville overskrive hinandens ændringer, som ville resultere i at kun et mod der havde noget at gøre med en specifik fil ville kunne virke på samme tid.

### 4.2.2 Ligning

Da vi nu ved at vi kan ændre på generations koden i spillet, kan vi kigge tilbage påhvad det egentlig er vi vil implementere.

Som tidligere nægt består *Minecraft* verdener af  $1\ m^3$  blokke i et tre dimensionels koordinatsystem. Koordinatsystemet er uendeligt bredt og langt, men har en begrænset højde på 256 blokke. Vores funktion kræver en del tilpasninger for at blive illustreret vel i dette koordinatsystem.

Den første ændring der skal gøres ved funktinen er at bytte om på nogle akser, da coordinat systemet i *Minecraft* bryger *Y* aksen som højde, i stedet

for  $Z$  aksen som bruges i vores ligning. Den nye funktio ser altså så ledes ud

$$y = \cos(x^2) + \sin(z^2)$$

Denne funtion bruger  $XZ$  planet som baggrund for at regne hjden  $Y$  ud, på samme måde som *Minecraft* gør det.

Det næste problem ved formlen der takles er at sinus og cosinus funktionerne bægge giver et tal mellem -1 og 1. Altså vil formlen give et resultat mellem -2 og 2 afhængigt af inputtet, vi kan kun generere blokke mellem 0 og 256.

Hvad jeg har valgt at gøre ved detter at lægge 64 til det endelige resultat af funktionen, hvilket gør at resultatet nu variere mellem 62 og 66, et niveau vi kan placere blokke i uden problemer.

$$y = (\cos(x^2) + \sin(z^2)) + 64$$

Den næste problemstilling der skal ændres på er at resultatet af vores funktion kun variere med 4, på trods af at det er helt accepablet, fungere det ikke helt godt for *Minecraft*, da vi normalt vil have at vores verden variere mere end kun 4 blokke op og ned, det giver et meget flat landskab.

Derfor ganger vi resultatet af sinus og cosinus funktionerne med 32, som giver en variation af 128 blokke ( $32 * 2$  op og  $32 * 2$  ned). Koblet med ændringen om at flytte hele kurven 64 blokke op, betyder det at kurven nu kan gå fra 0 til 128 i  $Y$  aksen.

$$y = ((\cos(x^2) + \sin(z^2)) * 32) + 64$$

Da vi nu kender den ligning vi vil implementere i *Minecraft*, vil vi fortsætte med den rent faktiske implementation af vores funktion i spillet.

## 4.3 Implementering

### 4.3.1 Iteration 1

Det første trin i at implementere vores generation ind i spillet er at definere et mod. Et mod har to hoved komponenter der krves for at spillet kan genkende det, og kører det.

Listing 4.1: SRPMod.java

```
1 package dk.philiphansen.srpmod;  
2
```

```

3 import net.minecraft.block.Block;
4 import cpw.mods.fml.common.Mod;
5
6 @Mod(modid = SRPMod.MODID, version = SRPMod.VERSION)
7 public class SRPMod
8 {
9     public static final String MODID = "SRPmod";
10    public static final String VERSION = "1.0";
11 }

```

---

Denne fil findes af minecraft under opstart, og kørers da den har `@mod` annotationen. Filen gør ikke noget da der ikke rent faktisk er noget kode at eksekvere, bare en fil der definerer at det *mod* findes.

Listing 4.2: *mcmod.info*

```

1 [
2 {
3     "modid": "SRPmod",
4     "name": "SRP Mod",
5     "description": "World generation mod for SRP.",
6     "version": "1.0",
7     "mcversion": "1.6.4",
8     "url": "",
9     "updateUrl": "",
10    "authors": ["Philip Hansen"],
11    "credits": "",
12    "logoFile": "",
13    "screenshots": [],
14    "dependencies": []
15 }
16 ]

```

---

*mcmod.info* er en fil der stræng taget ikke kræves, men den giver spillet en del informationer om vores *mod*, som vises i spillet. Denne information er der mest af alt for at hjælpe spilleren med at finde ud af hvad der er installeret. Se Appendix E for visning af *mcmod.info* i spillet.

### 4.3.2 Iteration 2

Da vi nu har et fungerende mod, der kan kørers i spillet, kan vi fortsætte med implementeringen af den egentlige funktionalitet.

For at holde styr på koden, laver vi først en ny fil til at holde vores generations kode, den kaldes *SRPWorldGenerator.java*, derefter skal denne

fil registeres som en generator, via *forge*, det gøres med følgene kode

Listing 4.3: SRPMod.java

```
1 public static SRPWorldGenerator worldGen = new SRPWorldGenerator();  
2  
3 @EventHandler  
4 public void init(FMLInitializationEvent event)  
5 {  
6     GameRegistry.registerWorldGenerator(worldGen);  
7 }  
8 }
```

For at teste at det virker som ønsket, har jeg kopieret noget test kode, der generere træ blokke, rund omkring i verdenen.

Listing 4.4: SRPWorldGenerator.java

```
1 package dk.philiphansen.srpmod;  
2  
3 import java.util.Random;  
4  
5 import net.minecraft.world.World;  
6 import net.minecraft.world.chunk.IChunkProvider;  
7 import cpw.mods.fml.common.IWorldGenerator;  
8  
9 public class SRPWorldGenerator implements IWorldGenerator {  
10  
11     @Override  
12     public void generate(Random random, int chunkX, int chunkZ, World world,  
13         IChunkProvider chunkGenerator, IChunkProvider chunkProvider) {  
14         world.setBlock(chunkX*16 + random.nextInt(16), 100, chunkZ*16 +  
15             random.nextInt(16), 5);  
16     }  
17 }
```

Da funktionen *generate* kaldes med *X*, *Y* værdien af den chunk der genereres, kan denne ganges med 16 (da enhver chunk er 16 \* 16 blokke, dette giver hjørnet på den givne chunk. Ved derefter at ligge et random tal mellem 0 og 16 til, får vi en blok med *Y* koordinat 100, og et random *X*, *Y* koordinat i hver chunk. Se Appendix F.

Med et stabilt grundlag for at generere blokke, fortsætter vi mod at generere den forskrevne funktion.

### 4.3.3 Iteration 3

Det næste skridt i mod at generere vores funktion er at placere en blok for hver punkt, i stedet for at placere en for hver chunk, da vores funktion vil kræve at vi udregner en  $Y$  værdi for hver punkt i  $XZ$  planet.

For at opnå dette mål bruges der to løkker, for at gå igennem hvert  $X$  og  $Z$  punkt i en chunk.

Listing 4.5: SRPWorldGenerator.java

```
1 for (int i = 1; i <= 16; i++) {  
2     for (int j = 1; j <= 16; j++) {  
3         world.setBlock(chunkX*16 + i, 64, chunkZ*16 + j, 5);  
4     }  
5 }
```

Med denne ændring i koden, bliver alle punkter med  $Y$  koordinat 64 sat til at være et stykke træ. Dette kan ses i Appendix G.

### 4.3.4 Iteration 4

Efter at have implementeret løkker i generationskoden, er det næste trin at lave en mere variable overflade, til det formål har jeg valgt at generere en cosinus kurve langs  $X$  aksen, der kun bruger den ene parameter.

Dette burde give en stor *bølge*, da kurven ikke ændre sig langs  $Z$  aksen.

Listing 4.6: SRPWorldGenerator.java

```
1 for (int i = 1; i <= 16; i++) {  
2     for (int j = 1; j <= 16; j++) {  
3         int coordinateX = chunkX*16 + i;  
4         int coordinateZ = chunkZ*16 + j;  
5         int coordinateY = (int)Math.round((Math.cos(coordinateX)) * 32) + 64;  
6         world.setBlock(coordinateX, coordinateY, coordinateZ, 5);  
7     }  
8 }
```

Løsningen fungere i teorien, men ved definitionen af min formel glemte jeg en vigtig detalje, sinus og cosinus funktioner har en bølgelængde af  $2 * \pi$ , det vil sige at min cosinus funktion går fra et  $Y$  koordinat på 96 til 32 på  $1 * \pi$ , eller lige over 3 blokke over  $X$  aksen. Dette resultere i noget der ikke så meget ligner en kurve, som en par punkter underligt forskudt.

Screenshot af denne funktion kan ses i Appendix H.

### 4.3.5 Iteration 5

For at komme frem til noget der ligner en rent faktisk bølge må vi altså give funktionen en periode der er lidt større end  $2 * \pi$ , dette gres ved at gange  $X$  koordinatet med en konstant der gør det mindre, før det bliver fodret ind i funktionen.

En konstant som 0.05 giver en periode på  $\frac{\pi}{0.05}$  eller 125 blokke, som giver noget der syneligt minder om en kurve.

Se Appendix I for visning af denne funktion.

### 4.3.6 Iteration 6

Med en fungerende cosinus kurve, er det næstte skridt at lave en funktion af to parametre. Som test har jeg valgt at generere funktionen

$$y = ((\cos(0.05 * x) + \sin(0.05 * z)) * 32) + 64$$

Resultatet er et flot bølgende landskab som kan ses i Appendix J, der begynder at ligne noget der kunne brugest at spille i.

### 4.3.7 Iteration 7

Den syvne og sidste iteration er implementering af den forskrevne funktion

$$y = ((\cos(0.025 * x^2) + \sin(0.025 * z^2)) * 32) + 64$$

Med den ændring at konstanten 0.05 er blevet ændret til 0.025, da vores funktion også er en andengrads funktion, og derfor har endnu større udslag end de tidlige funktioner.

Efter implementeringen af koden til denne funktion i vores *SRPWorldGenerator.java* fil, er koden til det endelige resultat som følger

Listing 4.7: SRPWorldGenerator.java

```
1 package dk.philiphansen.srpmod;
2
3 import java.util.Random;
4
5 import net.minecraft.world.World;
6 import net.minecraft.world.chunk.IChunkProvider;
7 import cpw.mods.fml.common.IWorldGenerator;
8
9 public class SRPWorldGenerator implements IWorldGenerator {
10     @Override
```

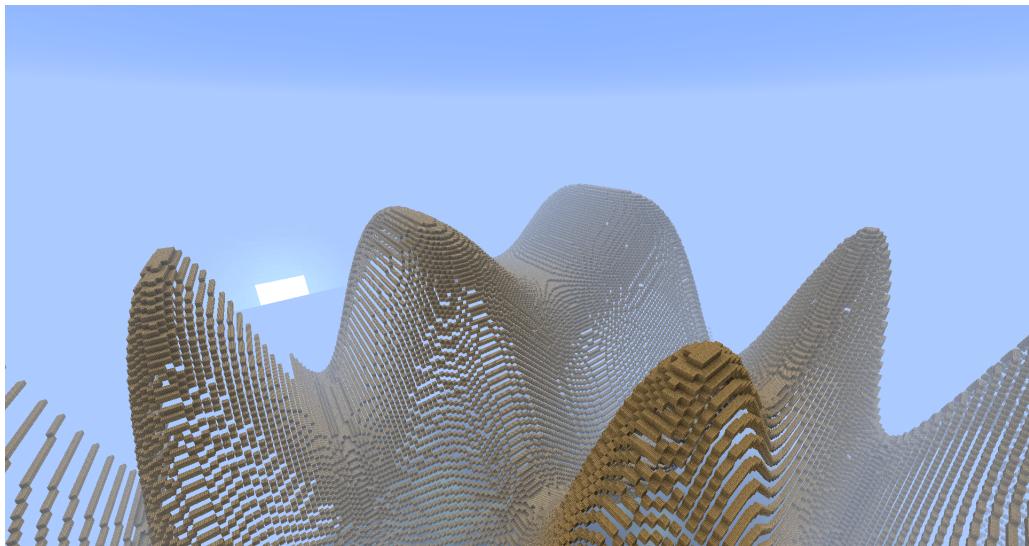
```

11     public void generate(Random random, int chunkX, int chunkZ, World world,
12                           IChunkProvider chunkGenerator, IChunkProvider chunkProvider) {
13         for (int i = 1; i <= 16; i++) {
14             for (int j = 1; j <= 16; j++) {
15                 int coordinateX = chunkX*16 + i;
16                 int coordinateZ = chunkZ*16 + j;
17                 int coordinateY = (int)Math.round((Math.cos(Math.pow(0.01 *
18                               coordinateX, 2)) + Math.sin(Math.pow(0.01 * coordinateZ, 2)
19                               )) * 32) + 64;
20                 world.setBlock(coordinateX, coordinateY, coordinateZ, 5);
21             }
22         }
23     }

```

---

Det endelige visuelle resultat kan ses her:



Figur 4.3:  $y = ((\cos(0.025 * x^2) + \sin(0.025 * z^2)) * 32) + 64$

På trods af at det måske kræver lidt kreativitet at se det, er det den samme funktion som blev plottet med *python* og *matplotlib*

## 4.4 Vurdering

På baggrund af mit arbejde med implementationen af denne funktion af to variabler, vil jeg vurdere at det er en god start, til måder at generere verdner på , men i sig selv er en funktion med to parametre ikke nok.

Som tidligere beskrevet, er en funktion med tre eller flere parametre nødvendig, i det mindste for at gøre spillet interesant. En funktion med kun to parametre  $X, Z$  vil altid generere på den samme måde, hvilket gør landskabet meget forudsigeligt.

Det ville være problematisks hvis spilleren kunne udregne hvor i jorden diamnter vil findes, og hvor skatte kan findes, ud fra koordinaterne på et punkt alene.

# Kapittel 5

## Perspektivering

Basering af hele verdner på baggrund af en relativt simpel ligning af denne art er for simplet, men der er god mulighed for brug af ligner af denne art i *Minecraft* alligevel.

Mindre genererede strukture, som huse, søer, farme og lignende kan sagtens genereres på baggrund af mindre komplicerede ligninger, som ikke har behov for mere end to koordinater.

Man kunne godt forstille sig at have en større algorytm der styrer hele generationen af verdenen, som kalder en funktion med et punkt på hvilket et hus skal genereres, hvorever en funktion af to parametre bruges til at generere selve strukturen.

# Kapittel 6

## Kilder

- [http://matplotlib.org/examples/mplot3d/surface3d\\_demo.html](http://matplotlib.org/examples/mplot3d/surface3d_demo.html)  
Som basis for plot.py, brugt til at lære hvordan man laver 3d grafer i matplotlib.
- [http://www.minecraftforge.net/wiki/Adding\\_World\\_Generation](http://www.minecraftforge.net/wiki/Adding_World_Generation)  
Brugt som baggrund for verden generering
- Matematik A - Forberedelsesmateriale - Mandag den 27. maj 2013,  
side 7 og 8  
Til forståelse og beskrivelse af funktioner af to parametre

# Appendix A

## plot.py

*plot.py* er funktionen jeg har brugt til at vise funktionen

$$y = \cos(x^2) + \sin(z^2)$$

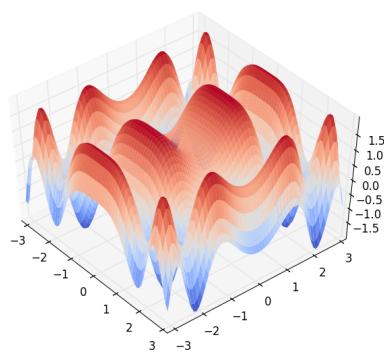
som en tre dimmensionel graf

Listing A.1: plot.py

```
1 from mpl_toolkits.mplot3d import Axes3D
2 from matplotlib import cm
3 import matplotlib.pyplot as plt
4 import numpy as np
5
6 fig = plt.figure()
7 ax = fig.gca(projection='3d')
8
9 X = np.arange(-np.pi, np.pi, 0.1)
10 Y = np.arange(-np.pi, np.pi, 0.1)
11 X, Y = np.meshgrid(X, Y)
12 Z = np.cos(np.square(X)) + np.sin(np.square(Y))
13
14 surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.coolwarm, linewidth
15 =0)
16 plt.show()
```

## Appendix B

plot1.png



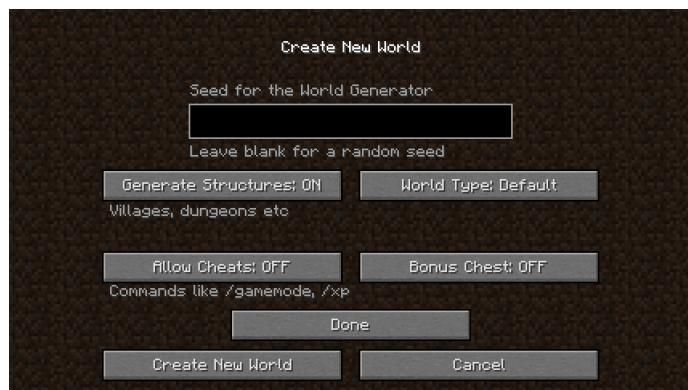
## Appendix C

screenshot1.png



## Appendix D

screenshot2.png



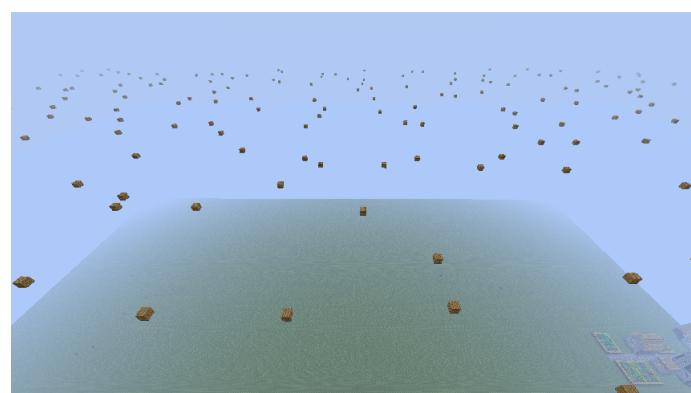
## Appendix E

### screenshot3.png



## Appendix F

screenshot4.png



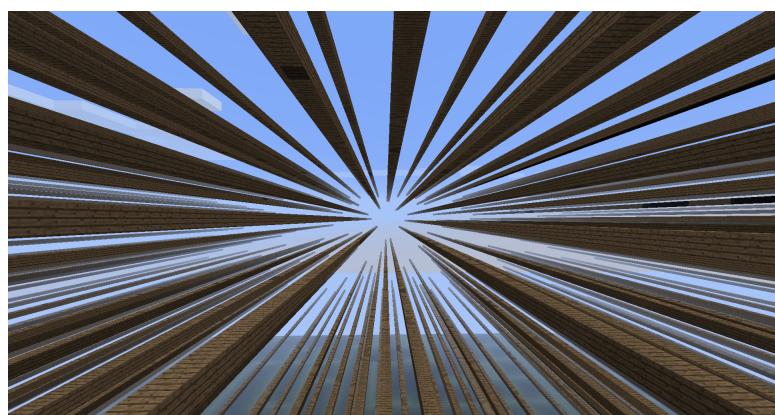
## Appendix G

screenshot5.png



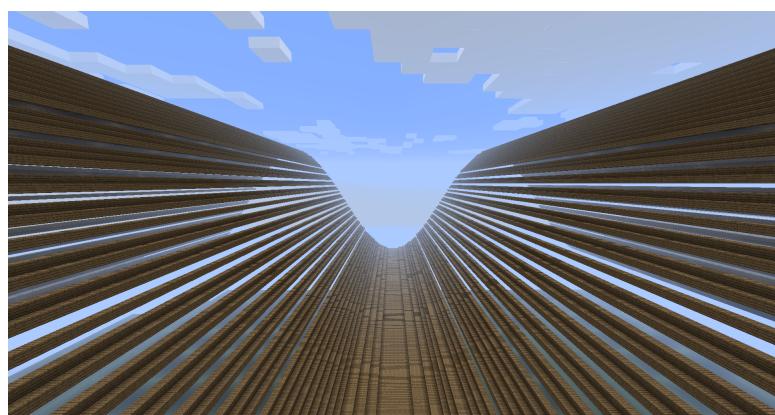
## Appendix H

screenshot6.png



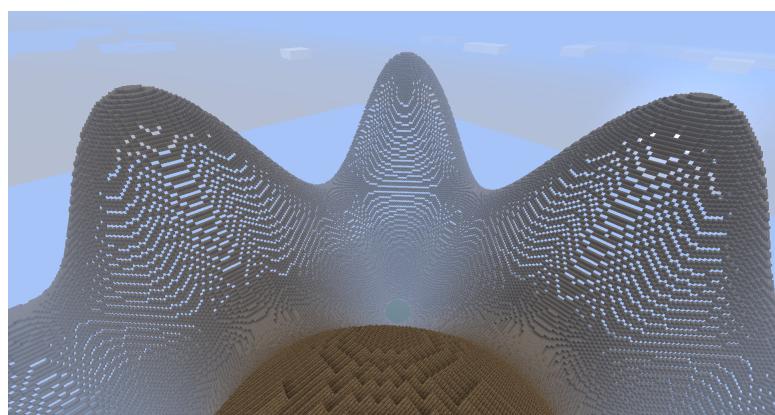
# Appendix I

screenshot7.png



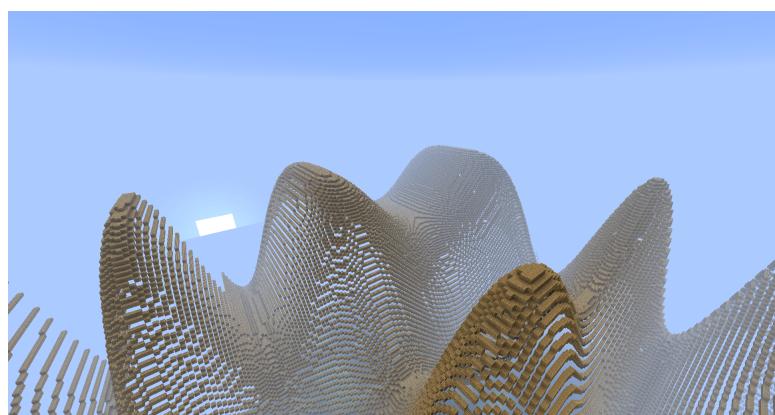
## Appendix J

screenshot8.png



## Appendix K

screenshot9.png



# Appendix L

## mcmod.info

Listing L.1: mcmod.info

```
1 [  
2 {  
3   "modid": "SRPmod",  
4   "name": "SRP Mod",  
5   "description": "World generation mod for SRP.",  
6   "version": "1.0",  
7   "mcversion": "1.6.4",  
8   "url": "",  
9   "updateUrl": "",  
10  "authors": ["Philip Hansen"],  
11  "credits": "",  
12  "logoFile": "",  
13  "screenshots": [],  
14  "dependencies": []  
15 }  
16 ]
```

# Appendix M

## SRPMod.java

Listing M.1: SRPMod.java

```
1 package dk.philiphansen.srpmod;
2
3 import net.minecraft.block.Block;
4 import cpw.mods.fml.common.Mod;
5 import cpw.mods.fml.common.Mod.EventHandler;
6 import cpw.mods.fml.common.event.FMLInitializationEvent;
7 import cpw.mods.fml.common.registry.GameRegistry;
8
9 @Mod(modid = SRPMod.MODID, version = SRPMod.VERSION)
10 public class SRPMod
11 {
12     public static final String MODID = "SRPmod";
13     public static final String VERSION = "1.0";
14     public static SRPWorldGenerator worldGen = new SRPWorldGenerator();
15
16     @EventHandler
17     public void init(FMLInitializationEvent event)
18     {
19         GameRegistry.registerWorldGenerator(worldGen);
20     }
21 }
```

# Appendix N

## SRPWorldGenerator.java

Listing N.1: SRPWorldGenerator.java

```
1 package dk.philiphansen.srpmod;
2
3 import java.util.Random;
4
5 import net.minecraft.world.World;
6 import net.minecraft.world.chunk.IChunkProvider;
7 import cpw.mods.fml.common.IWorldGenerator;
8
9 public class SRPWorldGenerator implements IWorldGenerator {
10
11     @Override
12     public void generate(Random random, int chunkX, int chunkZ, World world,
13             IChunkProvider chunkGenerator, IChunkProvider chunkProvider) {
14         for (int i = 1; i <= 16; i++) {
15             for (int j = 1; j <= 16; j++) {
16                 int coordinateX = chunkX*16 + i;
17                 int coordinateZ = chunkZ*16 + j;
18                 int coordinateY = (int)Math.round((Math.cos(Math.pow(0.025 *
19                     coordinateX, 2)) + Math.sin(Math.pow(0.025 * coordinateZ,
20                     2))) * 32) + 64;
21                 world.setBlock(coordinateX, coordinateY, coordinateZ, 5);
22             }
23         }
24     }
25 }
```