Visualisering af funktioner af to variable i Minecraft

Philip Peder Hansen

December 20, 2013

Contents

Figure 1: En *Minecraft* verden

0.1 Funktioner af to variable

0.1.1 Afbildning af funktioner af to variable

0.2 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 ild kugler 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 heder *redstone*, der til en vis grad fungere som ledninger. *Redstone* kan brugest 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 algorytmer, der resultere i et mønster der til nogen grad minder og den virkelige verden. Trær 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.

0.2.1 Generering af verdner

Algorytmerne 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 koordinated bruger minecraft også noget som kaldes et seed til at generere verdener, de genereres altså ud fra en mere 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.

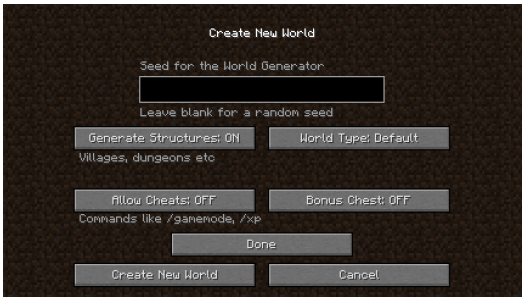

Figure 2: *Minecraft* verden genererings skærm

Forstil dig for eksempel dette scenarie, 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 interesante 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.

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.

0.2.2 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*.

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.

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 nænt 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 resutat

mellem -2 og 2 afhængligt at 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 resutatet af sinus og cosinus funktinerne 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.

0.2.3 Implementering

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 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 2: mcmod.info

```
[
{
  "modid": "SRPmod",
  "name": "SRP Mod",
  "description": "World generation mod for SRP.",
  "version": "1.0",
  "mcversion": "1.6.4",
  "url": "",
  "updateUrl": "",
  "authors": ["Philip Hansen"],
  "credits": "",
  "logoFile": "",
  "screenshots": [],
  "dependencies": []
}
]
```

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.

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 *SRP-WorldGenerator.java*, derefter skal denne fil registeres som en generator, via *forge*, det gøres med følgene kode

Figure 3: mcmod.info

Listing 3: SRPMod.java

```
    public static SRPWorldGenerator
worldGen = new SRPWorldGenerator();

```

```
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: 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, IChunkProvider chunkGenerator
, IChunkProvider chunkProvider) {
13          world.setBlock(
chunkX*16 + random.nextInt(16), 100,
 chunkZ*16 + random.nextInt(16), 5);
14      }
15  }
```

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 koordinated 100, og et random X, Y koordinat i hver chunk.

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

Iteration 3

0.2.4 Vurdering

0.3 Perspektivering

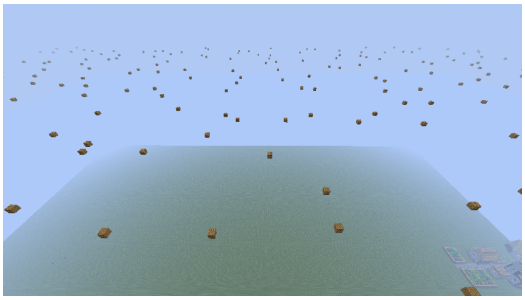

Figure 4: Første generations eksperiment