Perlin Noise och Brownian Motion – En guide.

Intro

I denna guide kommer jag att ta upp hur Brownian Motion fungerar i 2D och hur Perlin Noise fungerar 2D.

Jag kommer att gå igenom potentiella användnings områden och demonstrera hur den data vi kommer att framställa kan användas i ett program som kallas för WorldMachine.

Brownian Motion.

Brownian Motion är ett begrepp som har namngivits efter en Forskare som under senare delen av 1800-talet observerade att mycket små partiklar rör sig i kaotiska mönster.

Med hjälp utav dessa kaotiska mönster så kan man framställa information som går att använda till bland annat terräng generering till spel.

Principen bakom Brownian Motion är ganska simpel.

Tänk dig att du har en partikel i form utav en studsboll som ligger på ett bord med en massa andra studsbollar.

Om du sedan skulle skaka bordet alltså tillföra lite energi så skulle studsbollarna bumpa in i varandra och fara iväg i en till synes helt slumpmässig riktning för att sedan bumpa in i fler studsbollar för att byta riktning igen.

Om man sedan skulle välja ut en studsboll och doppa den i färg så att den lämnar spår efter sig så skulle vi få fram ett mycket slumpmässigt rörelsemönster.

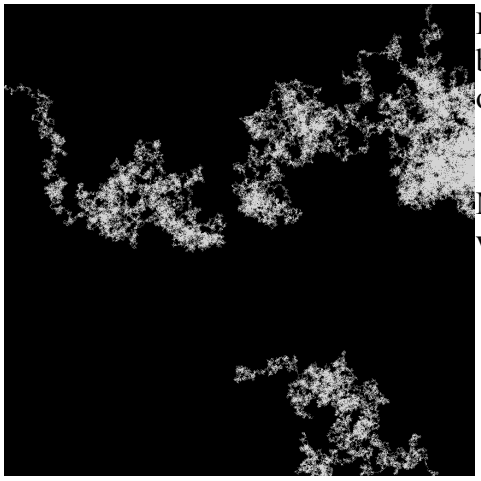
Att generera Brownian motion är relativt simpelt för någon med grundläggande programmerings kunskaper.

Pseudo-code:

```
//Loopa 100 000 gånger.
        for(i = 0; i < 100000;i++)
//Lägg till en ny slumpmässig riktning (x,y form på vektorn)
        Direction = Direction + RandomDirection();
//Lägg till den nya riktningen på vår position
        Position = Position + Normalize(Direction);
//Rita ut en pixel.
        SetPixel(Position,Red);
```

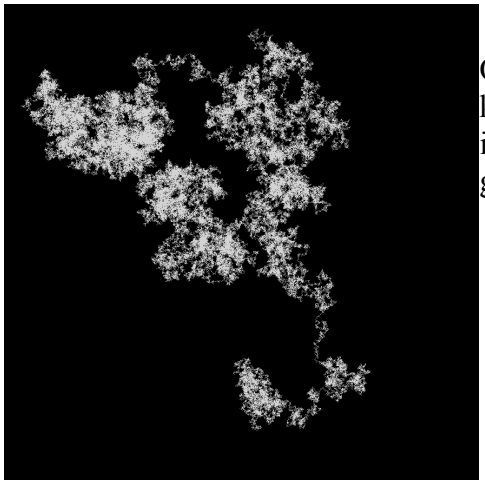
Det är inte mycket mer än så man behöver kunna för att komma igång med att använda detta för att generera mönster.

Ett tips är att man inte lägger på helt slumpmässiga riktningar. Om man viktar slumpen så att det är mindre chans att den åker utanför den specificerade bilden låt mig visa ett exempel.



På den här bilden kan man se att algoritmen har ritat utanför bilden och den lättaste lösningen på det är att bara klippa av och låta den fortsätta.

Men i många fall så är det inte bra att ha avklippta hörn på det viset.



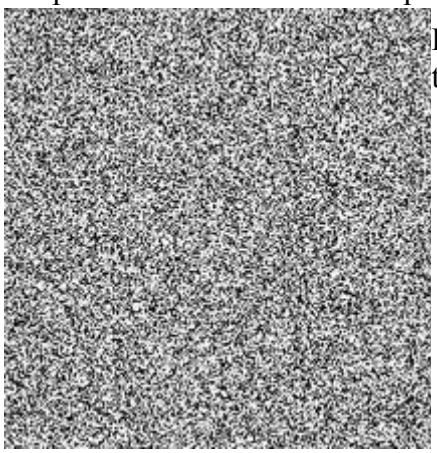
Om man istället låter slumpen bli lite svagare åt hållet som kanten är när den närmar sig så kan man på ett fint sätt tvinga in den till mitten utav bilden. Detta löser problemet på ett ganska så elegant sätt.

Brownian motions användningsområden i sig själv skulle kunna vara till exempel som mönster vid texturering eller för att styra till exempel väderfenomen. Tornadons rörelsemönster i spelet Epigenesis styrs utav samma principer.

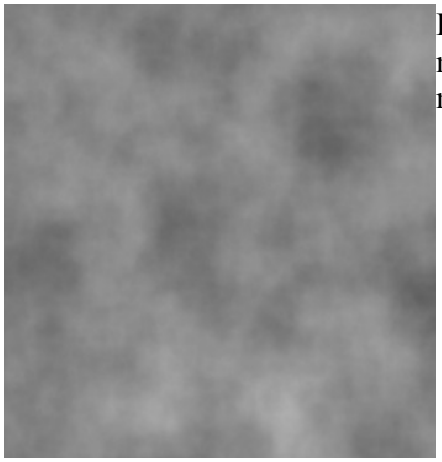
Perlin Noise.

Perlin Noise är ett sätt att generera ett brus som har en mjukare karaktär än vanligt brus igenom att lägga lager på lager utav samma brus med olika skala för att sedan ta medelvärdet utav dem.

Perlin Noise skapades utav Ken Perlin som sedan utvecklade en liknande algoritm som heter Simplex Noise som är bättre anpassa för applikationer som kräver högre hastighet.



Den vanligaste formen utav brus vi ser till exempel när vi slår på tvn på en ”tom kanal”.



Det här är perlin noise och består utav flera lager utav samma brus med olika skala och utgångspunkter som sedan har interpolerats i mellan.

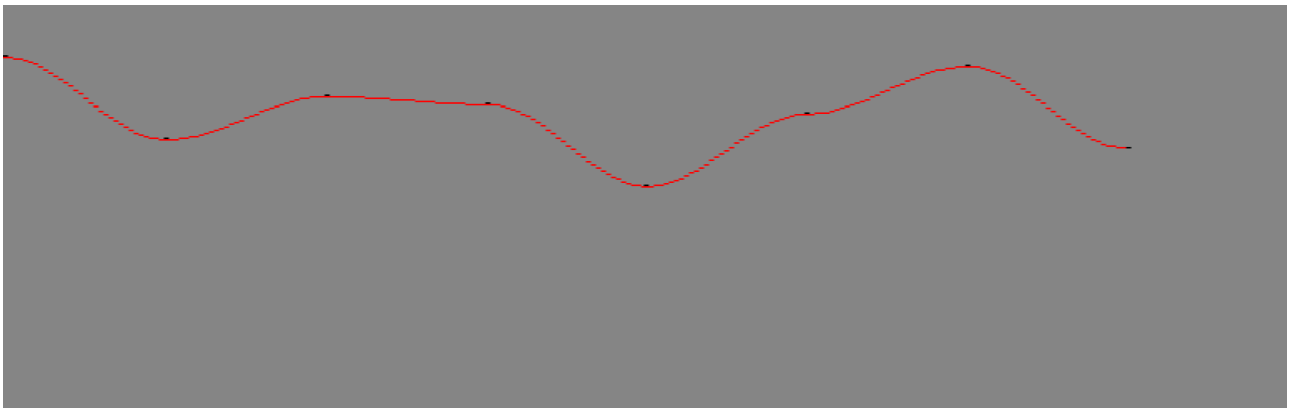
För att försöka mig på en liknelse förklaring.

Tänk dig att du har ett rutigt fiskenät uppsänt mellan ett par pinnar och sedan börjar du hänga vikter av olika vikt i alla utav fiskenätets rutor.

Fiskenätet kommer då att börja bölja sig men mer på de ställen där mer vikt hänger.

Sedan tar du ett par vikter i taget och knyter ihop dom så att ett större område påverkas utav den sammanlagda vikten och sedan låter du fiskenätets böjningar bestämma hur mörk/ljus en pixel i en bild ska vara. Så har du 2D Perlin noise.

Om man skulle se fiskenätet ifrån sidan alltså i en dimesion så skulle det se ut som något liknande och den röda kurvan är fiskenätet och de svarta pixlarna är vikter.



Några bitar kanske faller på plats med lite Pseudo-code**:**.

//2D array som ska hålla alla våra start värden

float[x,y] BV;

//Vi fyller en array full med flyttal mällan 0.0 och 1.0

void FillArray()

{

for(int x = 0;x < ImageW;x++)

for(int y = 0; < ImageH;y++)

BV[x,y] = RandomFloat(0,1); //Generate a random float from 0 to 1

}

//Här räknar vi ut ett medelvärde viktat efter vilka positioner som noiset befinner sig på

```
float SmoothNoise(int x, int y)
{
corners = (BV[x - 1, y - 1] + BV[x + 1, y - 1] + BV[x - 1, y + 1] + BV[x + 1, y + 1]) / 16;
sides   = (BV[x-1, y] + BV[x+1, y] + BV[x, y-1] + BV[x, y+1]) /  8;
center  =  BV[x, y] / 4;
return corners + sides + center;
}
```

//Cosinus interpolering gör så att bilden

```
float Cosine_Interpolate(float a, float b, float x)
{
ft = (x * 3.14159265);
f = (1 - Cos(ft)) * 0.5f;
return a * (1 - f) + b * f;
}
```

Jag valde att lämna alla typer i den här koden för att när jag läste den så var det en aning förvirrande vilka tal som skulle vara flyttal och vilka som skulle vara heltal.

Koden här går iallafal ut på att man räknar ut vilka punkter man vill ha SmoothNoise ifrån för att sedan Cosinus interpolera för att få fram ett mjukt medelvärde som sedan till sist blir ett prosent värde mellan 0 och 1 som kommer att beskriva färgen vid pixeln som den här funktionen körs för.

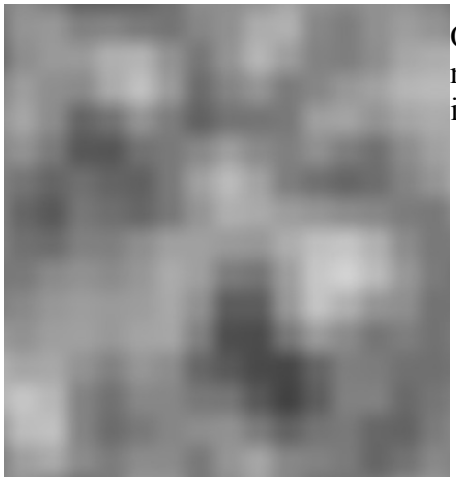
```
private float InterpolateSN(float x, float y)
{
     int integer_X    = (int)x;
     float fractional_X = x - integer_X;

     int integer_Y    = (int)y;
     float fractional_Y = y - integer_Y;

     float v1 = SmoothNoise(integer_X,     integer_Y);
     float v2 = SmoothNoise(integer_X + 1, integer_Y);
     float v3 = SmoothNoise(integer_X,     integer_Y + 1);
     float v4 = SmoothNoise(integer_X + 1, integer_Y + 1);

     float i1 = Cosine_Interpolate(v1 , v2 , fractional_X);
     float i2 = Cosine_Interpolate(v3 , v4 , fractional_X);

     return Cosine_Interpolate(i1, i2, fractional_Y);
}
```

Nu behöver vi bara köra våran InterpolateSN funktion på alla pixlar i bilden i olika lager
(Jag kommenterar på engelska i min källkod)

```
       for (int i = nI; i < Layers; i++)
       {
//Scale constant for that layer
                float layer = Math.Pow(2,i);

//Looping x and y for all pixels in the image
                for (int x = 0; x < w; x++)
                {
                    for (int y = 0; y < h; y++)
                    {
//Determain the procentage value to be looked up in our noise array
                        float nX = x / layer;
                        float nY = y / layer;

//Get the value from the noise algoritm
                        int col = (255 * InterpolateSN(nX, nY));

//Get the pixel that was located here earlier and do a simple avrage calculation for color.
//Could also cosinus interpolate here? :]

                        Color g = bmp.GetPixel(x, y);
                        int r = g.R;
                        int avgCol = (col + r) / 2;
                        Color cl = Color.FromArgb(avgCol, avgCol, avgCol);

                        SetPixel(x, y, cl);
                    }
                }
       }
```

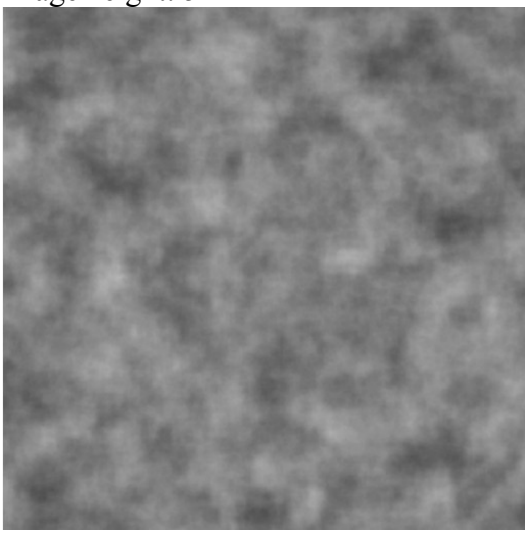
Vi loopar igenom alla pixlar i bilden för att ta reda på vad noise funktionen säger att färgen på varje pixel ska vara för att sedan med med 2^i där i är lagret/djupet vi vill kolla på för att sedan dividera våra pixel värden med det när vi vill ha fram värdet att skicka till noise funktionen.

Om det första lagret ser ut som något liknande det här nere så är allt som det ska.



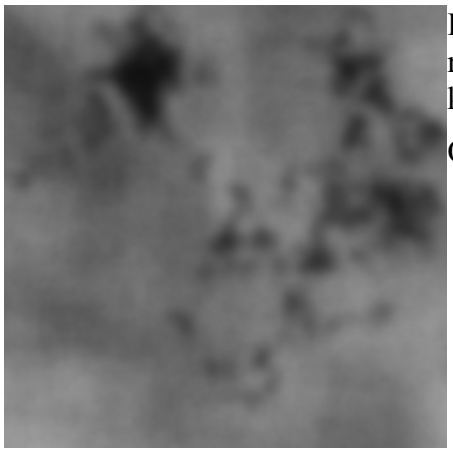
Om din bild börjar se ut så här när du kör alla lager så har du nog glömt att spara pixeldatan ifrån de tidigare bilderna och inte tagit medelvärdet utav dem.

Som referens har jag satt dessa parametrar för att generera denna bild:
Lager: 5
ImageWidth:512
ImageHeight: 512



World Machine – Bygga terräng

Jag har renderat fram en perlin noise och en brownian motion bild som jag lite snabbt kombinerade i Gimp (Ett bildredigerings program photoshop går bra det med)



De mörka områdena är där jag la in en Brownian Motion rendering med lite oskärpa på för att göra den lite mjukare i kanterna.

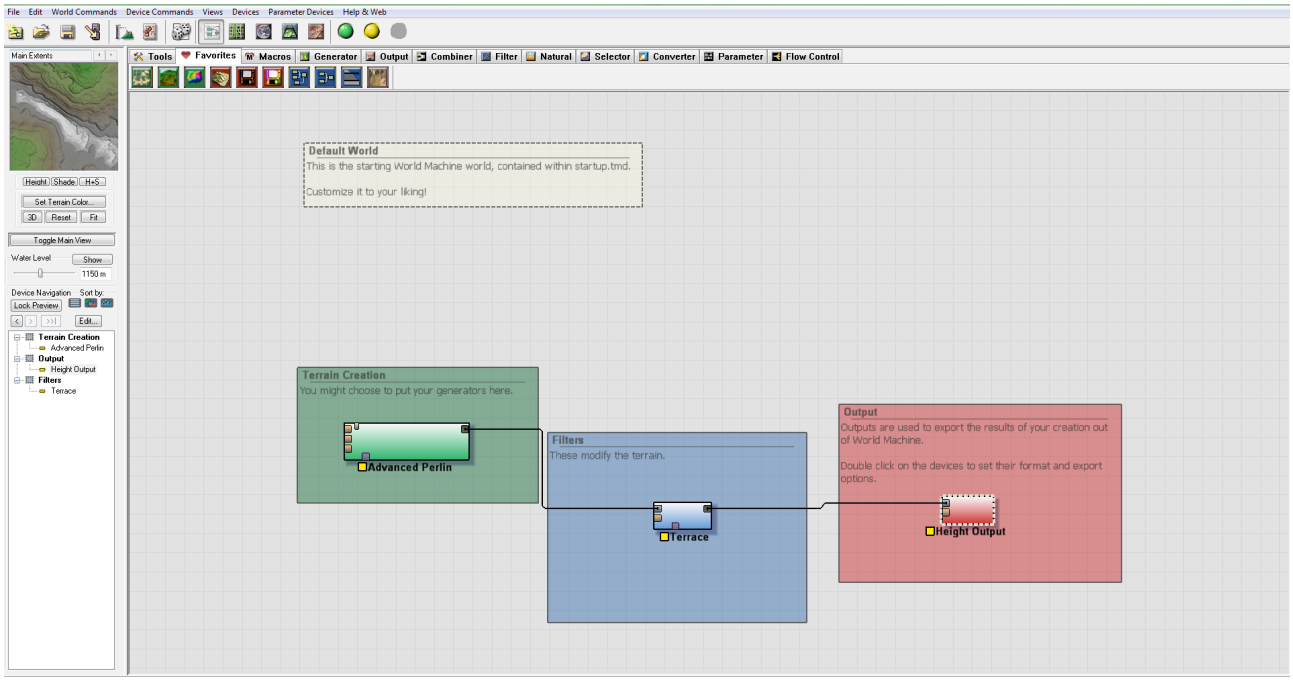
Och bakgrunden är perlin noise.

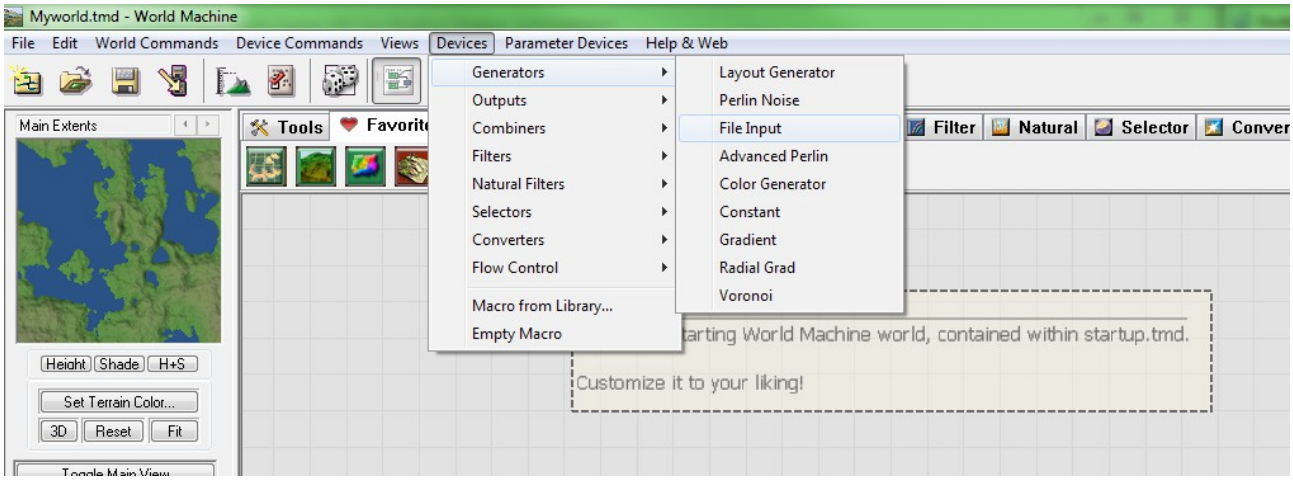
Jag laddade ner Worldmachine för att testa våra heightmaps:

http://www.world-machine.com/ Det finns en gratisversion som fungerar bra

Väl i worldmachine kommer du att se något liknande de har en bas scen uppsatt med en avancerad

perlin noise kan man tänka sig? Men vi vill ju inte använda deras så vi tar bort den och lägger till en from file generator (Markera noden och tryck delete)

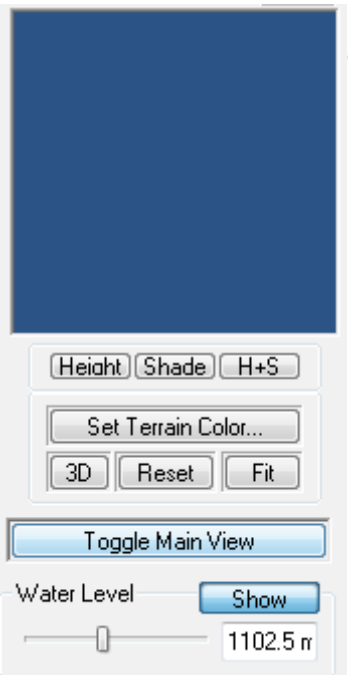




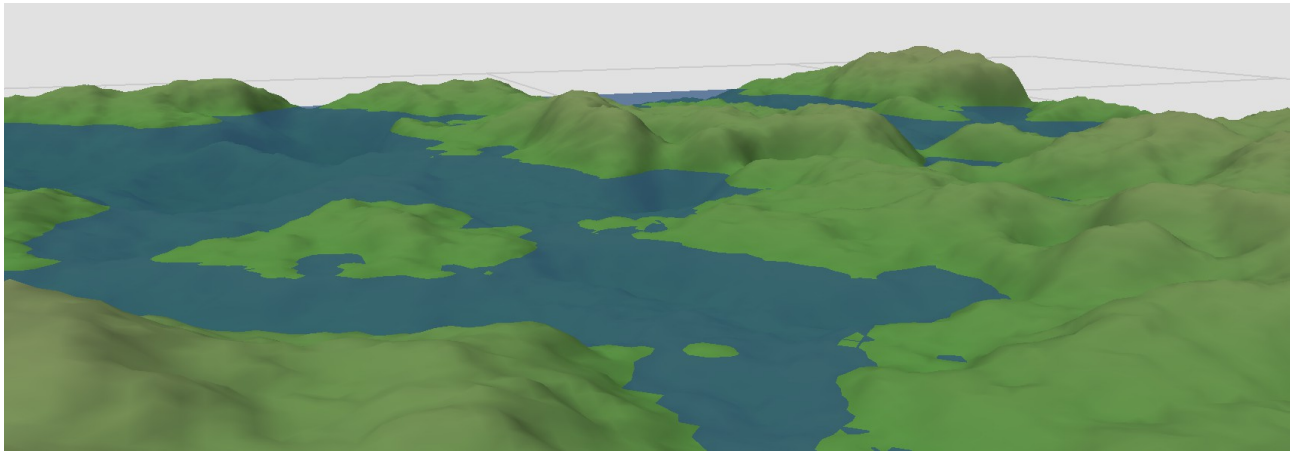
Under generators vill vi välja file input istället och lägga en sådan i vårt dra och släpp fönster där den förra generatorn låg.

Sedan är det bara att dubbelklicka på den och välja sin noise fil så borde vi börja se saker i preview bilden på sidan.

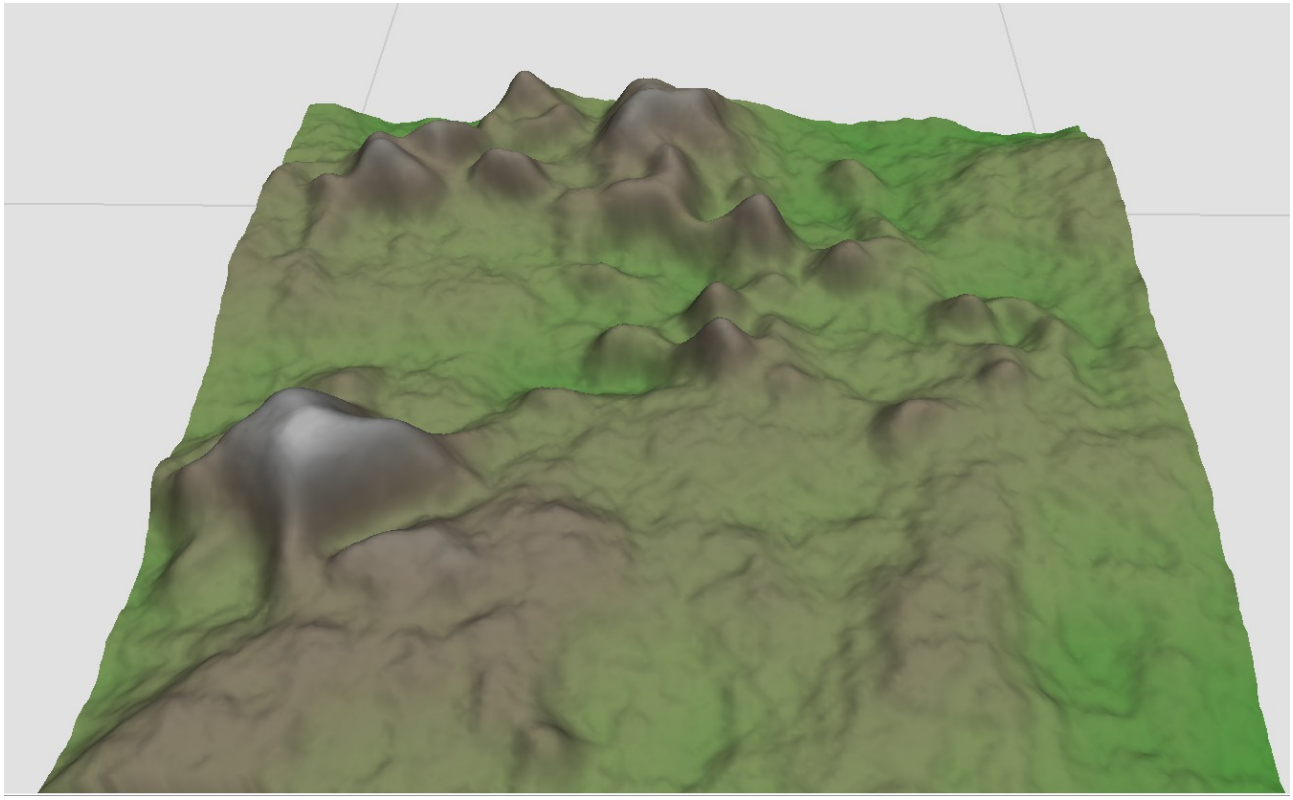
Jag ställde in vatten nivån tills dess att jag såg att det började hamna vatten i brownian motion skuggans lite djupare dalar. Glöm inte att trycka ner ”Show” knappen det fattade inte jag först.



Om man sedan går till 3D View igenom att trycka F8 så borde det synas ett fint landskap baserat på de olika noise vi har skapat.



Vattnet ligger där vi la vår lite mörkare brownian motion eftersom det är lägre ner.



Jag inverterade bildens färger och så fick vi en bergskedja där istället.

Det summerar det jag hade att försöka förklara mer läsning och referenser kommer nedan.

Browns original text ifrån 1828 http://sciweb.nybg.org/science2/pdfs/dws/Brownian.pdf

Pseudo-Kod och förklaringar utav Hugo Elias har hjälpt mig en hel del med att komma in i och använda mig utav Perlin Noise: http://freespace.virgin.net/hugo.elias/models/m_perlin.htm

Den här hemsidan av Matt Zucker förklarar mer matematiskt vad som försegår och har hjälpt mig förstå en del men jag hade svårigheter att förstå vissa delar som fick min första perlin noise generator att gå till skroten: http://webstaff.itn.liu.se/~stegu/TNM022-2005/perlinnoiselinks/perlin-noise-math-faq.html

Source koden till mina testprojekt finns här:

Brownian Motion:

https://www.dropbox.com/s/9b2wiz2b2zexk1z/BrownianMotion.7z

Perlin Noise:

https://www.dropbox.com/s/v2qpwb471529z2r/Perlin2_Source.7z