**Verslag PGT:**

**Procedurally Generated Terrain**

Naam: Jonathan Vercammen

Klas: 3ICT

Schooljaar: 2022-2023

Inhoud

[1 Inleiding 1](#_Toc119315169)

[2 Noisefuncties 1](#_Toc119315170)

[2.1 Introductie 1](#_Toc119315171)

[2.2 Noise 1](#_Toc119315172)

[2.3 Random Number Generator (RNG) 2](#_Toc119315173)

[2.4 Random noise 3](#_Toc119315174)

[2.5 Value noise 3](#_Toc119315175)

[2.5.1 Werking 4](#_Toc119315176)

[2.5.2 Tijdscomplexiteit 4](#_Toc119315177)

[2.5.3 Voor- en nadelen 4](#_Toc119315178)

[2.5.4 Toepassingen / Voorbeelden 4](#_Toc119315179)

[2.6 Simplex noise 4](#_Toc119315180)

[2.6.1 Werking 4](#_Toc119315181)

[2.6.2 Tijdscomplexiteit 4](#_Toc119315182)

[2.6.3 Voor- en nadelen 4](#_Toc119315183)

[2.6.4 Toepassingen / Voorbeelden 4](#_Toc119315184)

[2.7 Fractal noise (layered noise) 5](#_Toc119315185)

[2.8 Toespassingen 5](#_Toc119315186)

[3 Perlin noise 6](#_Toc119315187)

[3.1 Parameters / Begrippen 6](#_Toc119315188)

[3.2 Perlin vs Simplex noise 6](#_Toc119315189)

[3.3 6](#_Toc119315190)

[4 Procedurally Generated Terrain 6](#_Toc119315191)

[4.1 ... 6](#_Toc119315192)

[4.2 Problemen 6](#_Toc119315193)

[4.3 Optimalisatie 6](#_Toc119315194)

[5 Unity demo’s 6](#_Toc119315195)

[6 Besluit 6](#_Toc119315196)

[7 Codefragmenten 6](#_Toc119315197)

[8 Figurenlijst 7](#_Toc119315198)

[9 Bronvermelding 8](#_Toc119315199)

# Inleiding

~~Dit onderzoek gaat over de werking van procedurally generated terrain. Het eerste deel geeft een introductie over noisefuncties en bespreekt een aantal voorbeelden. Hoofdstuk 3 gaat dieper in op Perlin noise. Daarna volgt meer uitleg over PGT met verschillende voorbeelden in Unity....~~

Wat is het doel van dit project? Waarom Procedurally generated terrain?

# Noisefuncties

## Introductie

Willekeurigheid is overal in de wereld maar toch blijft het zo’n abstract begrip. Wanneer is iets random? Is het ontstaan van het universum of het gooien van een dobbeslteen random en hoe kunnen computers dit simuleren als ze enkel instructies kunnen volgen? Vooral de laatste vraag piekte de interesse van Ken Perlin, een professor van de New York-universiteit. Perlin vond namelijk dat computergegenereerde beelden niet organisch genoeg waren.

Tijdens het maken van de Walt Disney-film *Tron (1982)* bedacht hij een algoritme dat organische vormen kon maken. Enkele jaren later in 1997 kreeg Perlin een academische prijs van AMPAS, de Academy of Motion Picture Arts and Sciences, en kreeg het algroritme de toepasselijke naam Perlin Noise. (Wikipedia, 2022)

## Noise

Noise of ruis heeft verschillende betekenissen. Enerzijds duidt het op een storend geluid. Anderzijds betekent het een visuele storing. Denk bv. aan de sneeuw van een tv of het gekras van een radio. Voor een signaal betekent noise een willekeurige afwisseling in sterkte of snelheid.

Hoofdzakelijk bestaan er twee soorten noise namelijk value noise en gradient noise. Gradient noise heeft veel varianten. Enkele voorbeelden zijn Simplex noise, Perlin noise, simulation noise en wavelet noise. Er zijn ook speciale soorten noise zoals Worley noise of Fractal noise.

De volgende hoofdstukken gaan verder in op de meest gekende noisefuncties en hun werking.

## Random Number Generator (RNG)

Elke noisefunctie steunt op random getallen. Value noise bijvoorbeeld gebruikt deze getallen om rasterpunten op te vullen. Maar hoe kunnen computers getallen genereren? Dit is waar random number generators (RNG) van pas komen.

Computers genereren geen echte random getallen maar pseudorandom getallen. Dit zijn getallen die willekeurig lijken maar opgebouwd zijn door een vaste formule. Het grootste voordeel hiervan is dat het zeer snel werkt. Ook is er maar een kleine hoeveelheid geheugen nodig. Het nadeel hieraan is dat een formule altijd hetzelfde resultaat zal teruggeven. Om meer variatie te geven, maakt een RNG gebruik van een ‘seed’. Dit is een cijfer die bepaalt hoe de generator getallen berekent.

Een voorbeeld van een random number generator is de lineaire-congruentiegenerator. Deze heeft als formule: waarbij

|  |
| --- |
| X0­­­: seed / startwaarde |
| m: modulus, ofwel de rest na deling |
| a: vermenigvuldigingsfactor |
| c­­­: toename |

Hieronder een voorbeeld met parameters X0­­­=5, m=6, a=2 en c­­­=8.

Hier is meteen een probleem zichtbaar. Na voldoende iteraties ontstaat er zich een patroon. Parameterkeuze is bij deze generator heel belangrijk. Wanneer de uitkomst kleiner is dan tien dan zullen er in totaal maximum tien iteraties gebeuren zonder dubbele getallen. Daarom is het belangrijk om de parameters zo groot mogelijk te houden. Zo is de kans kleiner dat hetzelfde getal twee keer voorkomt.

Om een voorbeeld te geven, Java gebruikt voor haar Random()-methode volgende parameters: m= 2­­­­48, a= 25.214.903.917 en c­­­= 11. (Wikipedia, Linear congruential generator, 2022)

## Random noise

De simpelste vorm van noise is random noise. Random noise bestaat uit een n-dimensioneel raster waarin elk punt een willeukeurige waarde krijgt tussen nul en één, respectievelijk zwart en wit. In de figuur hiernaast is een voorbeeld te zien van random noise in twee dimensies.

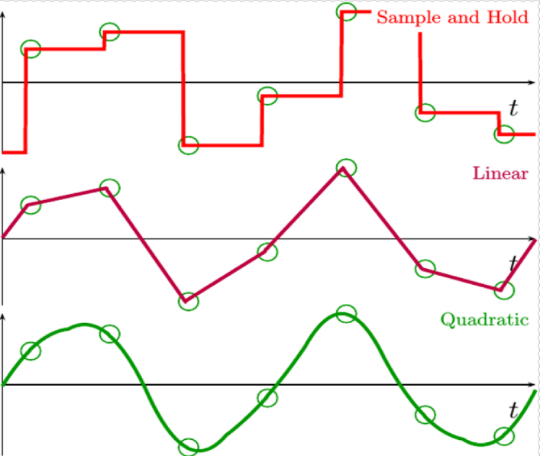
Het is duidelijk zichtbaar dat dit geen natuurlijk resultaat oplevert maar dat is ook niet het doel van random noise. Enkele toepassingen van random noise, beter bekent als white noise, zijn het maskeren van storende geluiden en in de geluidssector om versterkers te testen of audiosynthese. (Wikipedia, White noise, 2022)

Figuur 1 Een 2D raster met white noise

## Value noise

Net zoals random noise bestaat value noise uit een n-dimensioneel raster van willekeurige punten. Het verschil is dat value noise alle tussenwaardes interpoleert door te kijken naar de dichtstbijzijnde rasterpunten. In één dimensie betekent dit een interpolatie tussen de twee dichtstbijzijnde rasterpunten. In 2D, tussen de vier dichtstbijzijnde rasterpunten. Hiernaast staat een voorbeeld van value noise. Figuur 2

Figuur 2 Een 2D raster met value noise

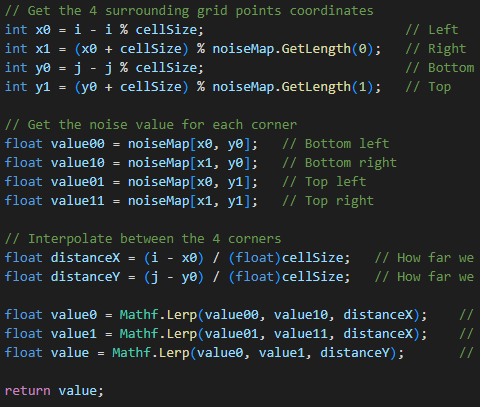
[](https://zipcpu.com/dsp/2018/03/30/quadratic.html)Er bestaan verschillende technieken om te interpoleren, meer hierover in volgend hoofdstuk. De meest gebruikte zijn linear en quadratisch. Lineair geeft snel resultaat maar ziet er vaak blokkerig uit. Quadratisch geeft het beste resultaat maar heeft veel rekenkracht nodig. Op onderstaande afbeelding is een voorbeeld zichtbaar van lineare en quadratische interpolatie. (Archer, 2011)

Figuur 3 Verschil tussen lineare en quadratische interpolatie

### Werking

Value noise start met een n-dimensioneel raster. In onderstaand voorbeeld staat de werking uitgelegd van value noise in twee dimensies met lineare interpolatie.

Stel eerst een 2D raster op en vul elk punt met een willekeurige waarde tussen nul en één Figuur 3. Voor elke waarde tussen deze punten, bereken de geïnterpoleerde waarde van de vier omliggende rasterpunten. Onderstaand codefragment toont een manier om deze waarde te berekenen voor een punt (i , j).

Het eerste blok code zoekt de coördinaten van de cel waarin het punt zich bevindt Figuur 4. Daarmee worden de waardes van elk hoekpunt opgevraagt Figuur 5. Vervolgens berekent de code de positie van het punt in de huidige cel en zet die om in procent Figuur 6. Ten slotte wordt interpolatie drie keer toegepast. Figuur 7

Codefragment 1 Interpolatie tussen twee punten i en j

De eerste keer voor de onderste twee punten, de tweede keer voor de bovenste twee punten en de laatste keer om verticaal te interpoleren.

Deze techniek heet ‘Bi-lineare interpolatie’.

|  |  |  |  |
| --- | --- | --- | --- |
| Figuur 4 Raster met vier punten | Figuur 5 Celcoördinaten zoeken | Figuur 6 Waardes van hoekpunten zoeken | Figuur 7 Positie in cel zoeken |
| Figuur 8 Bi-lineare interpolatie |  |  |  |

### Eigenschappen

Speed: 3.7 Value Noise alone is very fast, however when compounded with fBm it becomes

quite slow.

Memory: 10 Value Noise can be implemented to determine each pixel on the fly, without

needing to store anything in memory. Perlin Noise, as seen later, can also do this, but must

store a small permutation table and gradient table as well.

Quality: 6 - 8 Depending on the interpolation function, quality ranges from grainy to good.

### Toepassingen / Voorbeelden

<https://www.wallstreetmojo.com/interpolation/>

(2.3) <https://micsymposium.org/mics_2011_proceedings/mics2011_submission_30.pdf>

## Fractal noise (layered noise)

In de praktijk worden zuivere noisefuncties bijna niet gebruikt. Altijd meerdere “lagen” noise => fractal noise.

Fractal Brownian motion

https://www.provideocoalition.com/fractal-noise-advanced-analysis-of-after-effects-most-versatile-plugin/#:~:text=The%20basic%20concept%20of%20fractal,with%20increasing%20levels%20of%20detail.

## Toespassingen

Procedural generation, graphics, video games

# Perlin noise

(2.4) <https://micsymposium.org/mics_2011_proceedings/mics2011_submission_30.pdf>

<https://www.ronja-tutorials.com/post/026-perlin-noise/>

## Parameters / Begrippen

<https://docs.aws.amazon.com/lumberyard/latest/userguide/component-gradients-fastnoise.html>

## Simplex noise

<https://www.bit-101.com/blog/2021/07/perlin-vs-simplex/>

<https://en.wikipedia.org/wiki/Simplex_noise>

<https://en.wikipedia.org/wiki/Simplex>

<https://en.wikipedia.org/wiki/Hypercube>

(2.5) <https://micsymposium.org/mics_2011_proceedings/mics2011_submission_30.pdf>

## 

# Procedurally Generated Terrain

<https://www.youtube.com/watch?v=CSa5O6knuwI>

https://www.youtube.com/watch?v=fjZAgoxFKiQ

## ...

## Problemen

## Optimalisatie

# Unity demo’s

<https://noiseposti.ng/posts/2022-01-16-The-Perlin-Problem-Moving-Past-Square-Noise.html>

Unity noise functie niet goed

# Besluit

# Codefragmenten

[Codefragment 1 Interpolatie tussen twee punten i en j 4](file:///H:\Jonathan\School\3de%20hogeschool\POP\2223pop-jonathanvercammen\Verslag%20PGT.docx#_Toc119418483)

# Figurenlijst

[Figuur 1 Een 2D raster met white noise 3](file:///H:\Jonathan\School\3de%20hogeschool\POP\2223pop-jonathanvercammen\Verslag%20PGT.docx#_Toc119418465)

[Figuur 2 Een 2D raster met value noise 3](file:///H:\Jonathan\School\3de%20hogeschool\POP\2223pop-jonathanvercammen\Verslag%20PGT.docx#_Toc119418466)

[Figuur 3 Verschil tussen lineare en quadratische interpolatie 3](file:///H:\Jonathan\School\3de%20hogeschool\POP\2223pop-jonathanvercammen\Verslag%20PGT.docx#_Toc119418467)

[Figuur 4 Raster met vier punten 4](#_Toc119418468)

[Figuur 5 Celcoördinaten zoeken 4](#_Toc119418469)

[Figuur 6 Waardes van hoekpunten zoeken 4](#_Toc119418470)

[Figuur 7 Positie in cel zoeken 4](#_Toc119418471)

[Figuur 8 Bi-lineare interpolatie 4](#_Toc119418472)

# Bronvermelding

Archer, T. (2011). *Procedurally Generating Terrain.* Opgehaald van micsymposium: https://micsymposium.org/mics\_2011\_proceedings/mics2011\_submission\_30.pdf

Wikipedia. (2022, September 13). Opgehaald van White noise: https://en.wikipedia.org/wiki/White\_noise

Wikipedia. (2022, Octobre 23). Opgehaald van Linear congruential generator: https://en.wikipedia.org/wiki/Linear\_congruential\_generator

Wikipedia. (2022, October 14). *Perlin Noise*. Opgehaald van Wikipedia: https://en.wikipedia.org/wiki/Perlin\_noise

[[1]](#footnote-1)

1. Verslag template van Richtpunt Campus Hamme [↑](#footnote-ref-1)