**Verslag PGT:**

**Procedurally Generated Terrain**

Naam: Jonathan Vercammen

Klas: 3ICT

Schooljaar: 2022-2023

Inhoud

[1 Inleiding 1](#_Toc119761885)

[2 Noisefuncties 1](#_Toc119761886)

[2.1 Introductie 1](#_Toc119761887)

[2.2 Noise 1](#_Toc119761888)

[2.3 Random Number Generator (RNG) 2](#_Toc119761889)

[2.4 Random noise 3](#_Toc119761890)

[2.5 Value noise 3](#_Toc119761891)

[2.5.1 Werking 4](#_Toc119761892)

[2.5.2 Eigenschappen 5](#_Toc119761893)

[2.5.3 Voorbeelden 5](#_Toc119761894)

[2.6 Perlin noise 6](#_Toc119761895)

[2.6.1 Werking 6](#_Toc119761896)

[2.6.2 Eigenschappen 7](#_Toc119761897)

[2.6.3 Voorbeelden 7](#_Toc119761898)

[2.7 Simplex noise 8](#_Toc119761899)

[2.8 Toepassingen 9](#_Toc119761900)

[3 Fractal noise 10](#_Toc119761901)

[3.1 Parameters / Begrippen 10](#_Toc119761902)

[4 Procedurally Generated Terrain 10](#_Toc119761903)

[4.1 ... 10](#_Toc119761904)

[4.2 Problemen 10](#_Toc119761905)

[4.3 Optimalisatie 10](#_Toc119761906)

[5 Unity demo’s 10](#_Toc119761907)

[6 Besluit 11](#_Toc119761908)

[7 Codefragmenten 11](#_Toc119761909)

[8 Figurenlijst 12](#_Toc119761910)

[9 Bronvermelding 13](#_Toc119761911)

# 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.

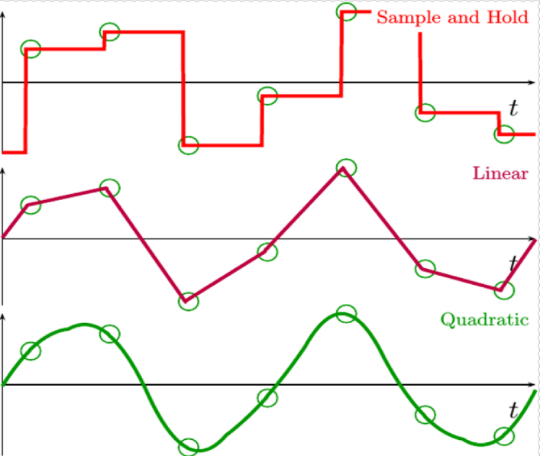
Het is duidelijk zichtbaar dat dit geen natuurlijk resultaat oplevert maar dat is ook niet het doel van random noise. Enkele toepassingen van random noise, beter bekent als white noise, zijn het maskeren van storende geluiden en in de geluidssector om versterkers te testen of audiosynthese. (Wikipedia,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

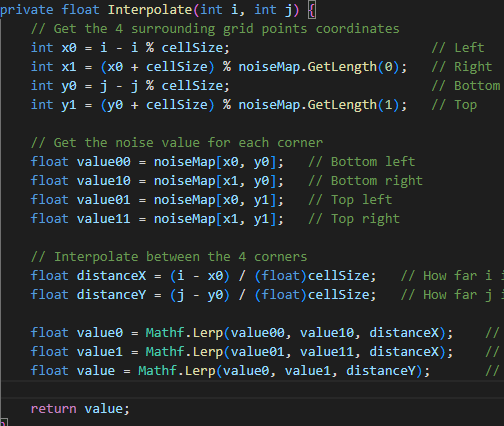
[](https://zipcpu.com/dsp/2018/03/30/quadratic.html)Er bestaan verschillende technieken om te interpoleren, meer hierover in volgend hoofdstuk. De meest gebruikte zijn linear en quadratisch. Lineair geeft snel resultaat maar ziet er vaak blokkerig uit. Quadratisch geeft het beste resultaat maar heeft 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

* Zeer snel voor één laag value noise maar schaalt niet goed met meerdere lagen.
* Hoeft noise-waarden niet op te slaan in het geheugen.
* Kwaliteit is sterk afhankelijk van de interpolatietechniek.

### Voorbeelden

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

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

## Perlin noise

Perlin noise is de meest gebruikte noisefunctie voor game development. Het is snel en geeft een natuurlijk resultaat.

Perlin noise is uitgevonden in 1982 door Ken Perlin. Later in 2001 verbeterde hij zijn algoritme dat nu gekend is als simplex noise, meer hierover in volgend hoodstuk.

Figuur 9 Een 2D raster met Perlin noise

In tegenstelling tot value noise gebruikt Perlin noise een raster van eenheidsvectoren i.p.v. punten. Deze vectoren duwen alle tussenliggende waardes weg van de oorsprong.

Onderstaande tekst legt de werking uit van Perlin noise in twee dimensies.

### Werking

Perlin noise werkt met twee lijsten: een lijst met vectoren in willekeurige richting en een lijst met random natuurlijke getallen, de permutatielijst. Deze lijst bestaat uit n getallen tussen 0 en n – 1. Dit vervangt de random number generator die nodig is bij value noise.

De eerste stap is dezelfde als bij value noise namelijk: stel een 2D raster op van willekeurige eenheidsvectoren, kies een punt en zoek zijn omliggende rasterpunten. Zoek daarna voor elk van deze punten hun bijhorende vector uit de vectorlijst met een hashfunctie Figuur 10. Een voorbeeld hiervan is: waarbij ‘p’ de permutatietabel is. Bereken daarna voor elk hoekpunt de afstandsvector naar het punt in de cel door het verschil te nemen van beide punten Figuur 11. De laatste stap is de interpolant berekenen van deze vier waardes. Dit gebeurt op dezelfde manier als bij value noise namelijk met lineare interpolatie.

|  |  |  |  |
| --- | --- | --- | --- |
| Figuur 10 gradient vectoren opzoeken | Figuur 11 afstandsvectoren berekenen |  |  |

### Eigenschappen

* Is een stuk trager dan value noise.
* Schaalt slecht bij meerdere dimensies
* Gebruikt een beetje geheugen om de vector- en permutatielijst op te slaan.
* Geeft een mooi resultaat.
* Afhankelijk van de implementatie kan Perlin noise herhalen.

### Voorbeelden

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

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

## Simplex noise

Simplex noise is de verbeterde versie van Perlin noise die ook is gemaakt door Perlin zelf. Het grootste probleem met Perlin noise is dat het heel traag werkt in meerdere dimensies. De oplossing die Perlin hiervoor heeft bedacht, is om te werken met een ander raster en weg te stappen van gewone vierkanten.

### Werking

In de meetkunde zijn er een aantal manieren om een multidimensionele figuur weer te geven. Eén daarvan is de hyperkubus. De meest bekende varianten zijn het vierkant in twee dimensies en de kubus in 3D.

Een tweede manier is met een simplex. Simplices zijn de famillie van de driehoek en de piramide. In onderstaande afbeeldingen staan de meest gekende hyperkubussen en simplices nog eens op een rijtje.

|  |  |  |
| --- | --- | --- |
| Figuur 12 2D hyperkubus, beter bekent als een vierkant | Figuur 13 3D hyperkubus, beter bekent als een kubus | Figuur 14 4D hyperkubus, beter bekent als een tesseract |

|  |  |  |
| --- | --- | --- |
| Figuur 15 2D simplex, ofwel een driehoek | Figuur 16 3D simplex, ofwel een tetraëder | Figuur 17 4D simplex, ofwel een pentachoron |

Een voordeel van simplices is dat er veel minder hoekpunten zijn om te berekenen. Bij Perlin noise is het aantal hoekpunten in n dimensies gelijk aan 2n. Bij simplex noise is dit maar n+1. Een bijkomend voordeel is dat lineare interpolatie volledig wegvalt. Simplex noise gebruikt een radiale dempingsfunctie om de tussenwaardes te berekenen.

De grootste moeilijkheid bij simplex noise is het vinden van de omliggende rasterpunten. De meest gebruikte techniek is om eerst alle rasterpunten te projecteren op een 2D vlak. De berekening van de omliggende hoekpuntent is vrij complex en zal niet worden besproken in dit verslag. Nadat alle hoekpunten bekend zijn, is de rest analoog met Perlin noise. Bereken daarna voor elk hoekpunt het dot product van de afstandsvector en de vector in het hoekpunt. De laatste stap gebruikt een dempingsfunctie om de uiteindelijk waarde te bekomen. Deze vervangt de interpolatie die nodig is bij Perlin noise en is ook een stuk sneller. (Archer, 2011)

<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>

### Eigenschappen

* Sneller dan Perlin noise
* Schaalt zeer goed met meerdere dimensies
* Gebruikt een beetje geheugen voor de permutatietabel en vectoren
* Geeft een beter resultaat dan Perlin noise

## Toepassingen

# Fractal noise

Fractal noise, of layered noise, is een veelgebruikte techniek om noise meer diepte te geven. De naam fractal noise komt van het meetkundig begrip ‘fractaal’. Een fractaal is een figuur die volledig zelfsymetrisch is. Dit wil zeggen dat op elke plaats een patroon zichtbaar is dat zichzelf continu herhaald tot in het oneindig. Het bekendste voorbeeld hiervan is de Mandelbrot, zie figuur 19.

|  |  |  |
| --- | --- | --- |
| [The Amazing World of Fractals | Be Loud! - A Yizzam Blog](https://blog.yizzam.com/the-amazing-world-of-fractals/)  Figuur 18 Voorbeeld van een fractaal | [How to Render a Fractal, Fast](https://blog.bede.io/how-to-render-a-fractal-fast/)  Figuur 19 Mandelbrot set | [Sierpiński triangle - Wikipedia](https://en.wikipedia.org/wiki/Sierpi%C5%84ski_triangle)  Figuur 20 Driehoek van Sierpiński |

......

## Parameters / Begrippen

TODO: foto’s toevoegen.

Fractal noise behaalt deze eigenschappen met een aantal parameters.

Het eerste belangrijk begrip is de amplitude. De amplitude van een functie geeft weer wat de maxima en minima van die functie zijn. Voor noise specifiek betekent dit hoe groot de noisewaarde op elk punt kan zijn.

Het tweede begrip is de frequentie. Deze parameter bepaalt hoe snel noisewaardes kunnen veranderen. Een hoge frequentie geeft veel meer details weer dan een lage frequentie.

De derde parameter is het aantal octaven. Neen, het zijn geen muzieknoten maar wel het aantal iteraties noise dat samengevoegd wordt. Hoe meer octaven, hoe meer details er zich zullen vormen. Let wel op want dit is een zware bewerking dus hou deze parameter klein.

De laatste twee parameters zijn iets complexer en hebben een verband met de voorgaande parameters. Het vierde begrip is lacunariteit en geeft weer hoe snel de frequentie stijgt per octaaf of per iteratie.

Het vijfde en laatste begrip is de persistentie en bepaalt hoeveel de amplitude verminderd per octaaf.

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

<https://web.archive.org/web/20220504014656/https://docs.aws.amazon.com/lumberyard/latest/userguide/component-gradients-fastnoise.html>

# 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#_Toc119761868)

[Figuur 2 Een 2D raster met value noise 3](file:///H:\Jonathan\School\3de%20hogeschool\POP\2223pop-jonathanvercammen\Verslag%20PGT.docx#_Toc119761869)

[Figuur 3 Verschil tussen lineare en quadratische interpolatie 3](file:///H:\Jonathan\School\3de%20hogeschool\POP\2223pop-jonathanvercammen\Verslag%20PGT.docx#_Toc119761870)

[Figuur 4 Raster met vier punten 4](#_Toc119761871)

[Figuur 5 Celcoördinaten zoeken 4](#_Toc119761872)

[Figuur 6 Waardes van hoekpunten zoeken 4](#_Toc119761873)

[Figuur 7 Positie in cel zoeken 4](#_Toc119761874)

[Figuur 8 Bi-lineare interpolatie 4](#_Toc119761875)

[Figuur 9 Een 2D raster met Perlin noise 6](file:///H:\Jonathan\School\3de%20hogeschool\POP\2223pop-jonathanvercammen\Verslag%20PGT.docx#_Toc119761876)

[Figuur 10 gradient vectoren opzoeken 6](#_Toc119761877)

[Figuur 11 afstandsvectoren berekenen 6](#_Toc119761878)

[Figuur 12 2D hyperkubus, beter bekent als een vierkant 8](#_Toc119761879)

[Figuur 13 3D hyperkubus, beter bekent als een kubus 8](#_Toc119761880)

[Figuur 14 4D hyperkubus, beter bekent als een tesseract 8](#_Toc119761881)

[Figuur 15 2D simplex, ofwel een driehoek 8](#_Toc119761882)

[Figuur 16 3D simplex, ofwel een tetraëder 8](#_Toc119761883)

[Figuur 17 4D simplex, ofwel een pentachoron 8](#_Toc119761884)

# 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

Wikitionary. (2022). Opgehaald van https://en.wiktionary.org/wiki

[[1]](#footnote-1)

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