Hoogtekaart

De eerste stap in het generen van onze wereld is een initiële hoogtekaart maken. Daarmee bedoelen wij een representatie die voor elke plek de hoogte aangeeft. De bedoeling is dat deze hoogtekaart later door andere modules kan worden aangepast, de rivier module zou bijvoorbeeld een pad kunnen weg eroderen.

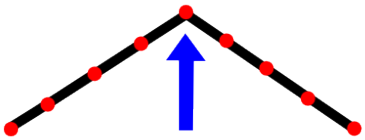
Om de hoogtekaart te generen hebben we besloten een variant van het midpoint-displacement algoritme, het diamond-square algoritme. Dit algoritme is een verbeterde variant van het midpoint-displacement algoritme en genereert net als het midpoint-displacement algoritme een (semi-)fractal. De hoogtekaart die gegenereerd wordt heeft dus een vergelijkbare opbouw op elke schaal. Dit algoritme wordt daarom vaak gebruikt voor het genereren van een terrein aangezien een landschap op grote schaal (een berg) grofweg dezelfde vorm heeft als op kleine schaal (een rots).

Figuur 1

Om het diamond-square algoritme voor een 2 dimensionale kaart uit te leggen is het handig om eerst het midpoint-displacement algoritme in 1 dimensie uit te leggen, aangezien het diamond-square algoritme een verbetering voor een 2 dimensionale kaart is van het midpoint-displacement algoritme.

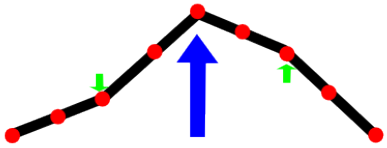
Het midpoint-displacement algoritme in 1 dimensie

Het midpoint-displacement algoritme werkt in 1 dimensie op een lijst van $2^n + 1$ punten, waarvan word aangenomen dat ze op een lijn allemaal even ver van elkaar af liggen, alle punten worden op een hoogte van 0 geplaatst. De startsituatie wordt in *figuur 1* grafisch weergegeven. De 9 punten ($2^3 + 1$) zijn in rood afgebeeld.

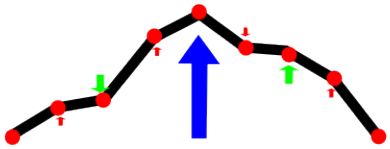


Figuur 2

Vervolgens wordt het middelste punt met een willekeurige hoeveelheid (uniform verdeeld) naar omhoog of omlaag verplaatst. In *figuur 2* is te zien hoe het middelste punt naar boven is verplaatst. Alle punten die nog geen waarde hebben gekregen bewegen mee alsof een punt op een touw omhooggetrokken wordt.



Figuur 3



Figuur 4

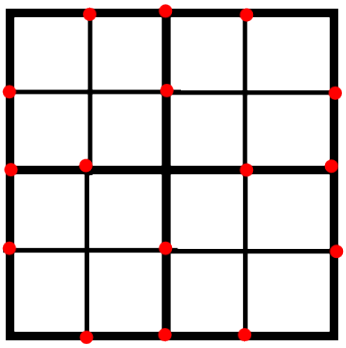
Figuur 5

De lijst wordt in deze fase in twee aparte lijsten opgedeeld, gesplitst op het middelste punt dat zojuist willekeurig bewogen is. Van beide van deze lijsten wordt weer het middelpunt genomen en willekeurig verplaatst (zie *figuur 3*). Nu is de verdeling van de afstand echter in een kleiner gebied, bijvoorbeeld tussen -0.5 en 0.5 in plaats van -1.0 en 1.0 bij het eerste middelpunt. Hoe kleiner de lijst wordt, hoe minder groot dit gebied ook zal zijn.. Hier wordt later meer over uitgelegd. Het is belangrijk om op te merken dat hoewel de hoogte in de figuren wordt aangegeven met een verplaatsing omhoog of omlaag, dit in het algoritme slechts een eigenschap van een punt is. Een punt zou dus ook niet loodrecht van de deellijst af verplaatsen maar nog steeds naar boven of beneden.

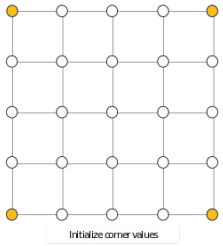
Ten slotte worden ook deze beide deellijsten weer opgedeeld en worden de laatste middelpunten willekeurig verplaatst (nog minder deze keer), zoals te zien is in *figuur 4*. Nu zijn er geen deellijsten meer te maken en is het algoritme klaar. Het eindresultaat is in *figuur 5* te zien.

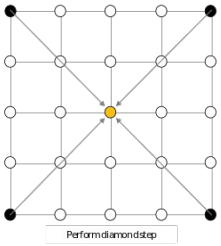
Het diamond-square algoritme

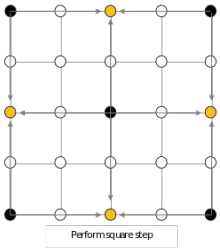
Het standaard midpoint-displacement algoritme werkt goed in 1 dimensie, maar bij 2 of meer dimensies komt het sterk tekort. Omdat bij midpoint-displacement alleen het middelste punt met een willekeurige afstand wordt verplaatst, worden veel hoekpunten overgeslagen. In *figuur 6* zijn ter illustratie in rood de punten weergegeven die bij het standaard midpoint-displacement algoritme niet willekeurig worden verplaatst in de eerste 2 verplaatsrondes. Het diamond-square algoritme lost dit probleem op door een extra fase aan elke ronde toe te voegen. Na deze fase zijn de aangepaste punten gearrangeerd in een rooster van vierkanten, deze stap wordt daarom de *square step* genoemd. Bij de stap waar de middelpunten verplaatst worden zijn de aangepaste punten juist gearrangeerd in een rooster van 90 graden geroteerde vierkanten die diamanten worden genoemd. Deze stap heet dan ook de *diamond step*. Hiervan komt ook de naam van het diamond-square algoritme. In *figuur 7* tot *10* – een illustratie van Wikipedia – zijn de stappen van het diamond-square algoritme op een vijf bij vijf rooster weergegeven. In *figuur 8* en *10* zijn de *diamond steps* te zien; in *figuur 9* en *11* de *square steps*.

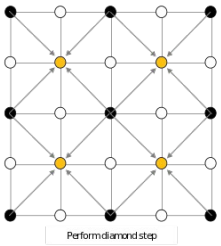


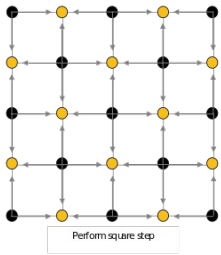
Figuur 6











Figuur 7 *Figuur 8* *Figuur 9* *Figuur 10* *Figuur 11*

Bij het diamond-square algoritme wordt het bereik van de willekeurig nummers vermindert hoe kleiner de sublijsten worden. Over het algemeen wordt het na elke ronde gedeeld door 2^h. Het bereik van het willekeurige nummer in ronde n is dus standaardbereik $* 2^{-h * n}$.

Alle punten op het behandelde rooster van het diamond-square algoritme worden 1 keer verplaatst. Het verplaatsen van punten in zowel de *square step* als in de *diamond step* is O(1). Het totale algoritme is dus *O(1 * aantal punten) = O(aantal punten)*.

Implementatie

Ter illustratie heb ik een simpele implementatie van het diamond-square algoritme in C++11 geschreven. Dit programma schrijft een hoogtekaart naar een zwart-wit PNG-illustratie bestand dat met een gewone foto viewer bekeken kan worden. Dit programma gebruikt de lodepng library om naar PNG-bestanden te schrijven en is te vinden op: http://paste.hydra.ws/p/o6R- .

- Marien Raat, 11 september 2015