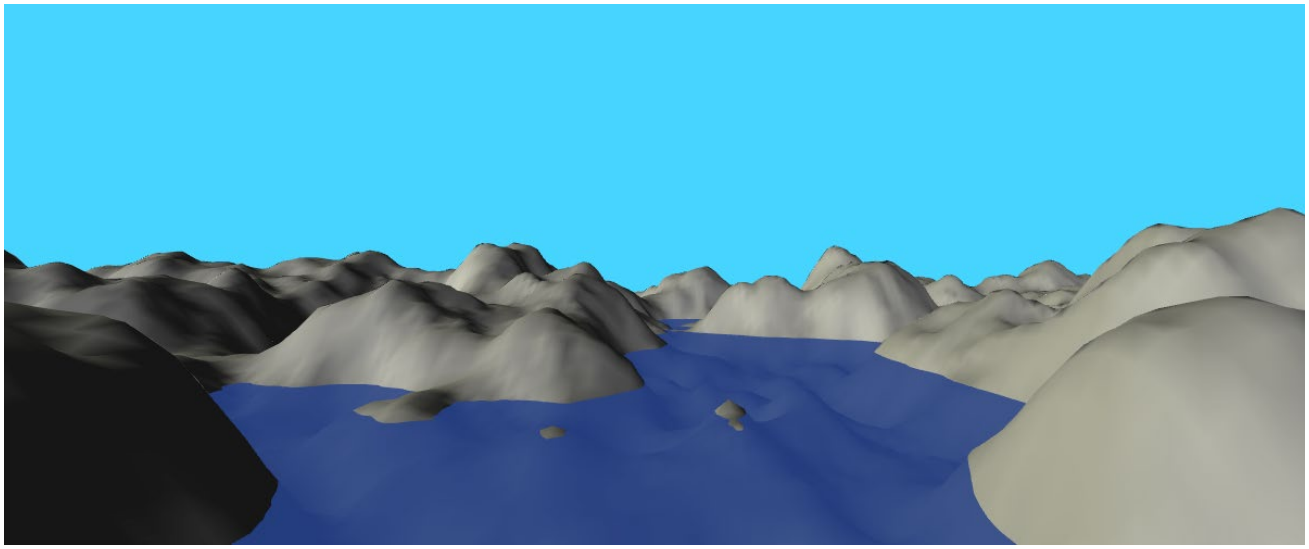


Kuva 2. Valmis maisema.

Maisemia Perlin-kohinalla

Peleissä, demoissa ja elokuvissa tarvitaan usein vaihtelevia, uskottavalta näyttäviä maastoja vuorineen ja laaksoineen. Kun maisemia luodaan algoritmien avulla, käytetään tyypillisesti apuna erilaisia kohinafunktioita. Eräs näistä on Perlin-kohina, joka tuottaa visuaalisesti miellyttävämpiä lopputuloksia kuin puhtaat satunnaisluvut. Tässä artikkelissa tehdään Processingilla pieni reaaliaikainen 3D-vuoristo.

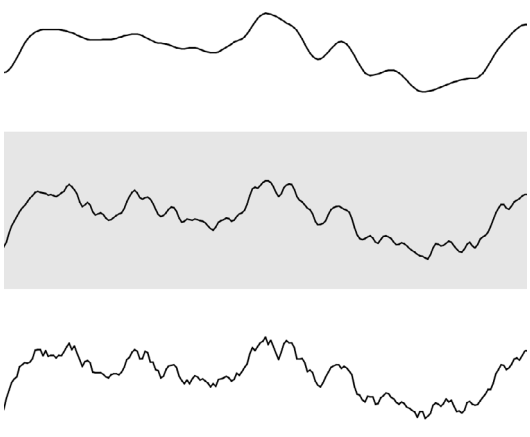
Teksti ja kuvat: Markku Reunanen

Perlin-kohinan perusteet

Ken Perlin kehitteli mukaansa nimetyn kohinafunktion 1980-luvun alkuvuosina tietokonegrafiikan ja -animaation tarpeisiin. Perlin-kohinaa on kolmenkymmenen vuotensa aikana käytetty varsin monenlaisiin tarkoituksiin, kuten esimerkiksi vuoristojen, pilvien, savun ja pintakuvioiden ohjelmalliseen luomiseen. Tavalliseen kohinaan verrattuna Perlin-kohina tuottaa luonnollisemmalta näyttävää grafiikkaa, jossa on samaan aikaan sekä suuria muotoja että pieniä yksityiskohtia.

Perlin-kohinan kantava idea on, että se koostuu useista satunnaisluvuista interpoloiduista niin kutsutuista oktaaveista, joista alimmilla (ensimmäisillä) on suuri amplitudi, mutta pieni taajuus. Tyypillisessä toteutuksessa jokaisessa seuraavassa oktaavissa taajuus kaksinkertaistetaan ja amplitudi puolitetaan. Alimmat oktaavit tuottavat siten pehmeitä suuria muotoja ja ylimmät teräviä nopeita muutoksia.

Kaksi keskeisintä muunneltavaa parametria ovat oktaavien kokonaismäärä sekä amplitudi. Kuvassa 1 näkyy sama kohina kahdella, neljällä ja kahdeksalla oktaavilla. Oktaaveja lisäämällä mukaan saa aina vain pienempiä yksityiskohtia, vaikka profiilin yleinen muoto säilyykin suunnilleen ennallaan. Amplitudi puolestaan määrittää sen, kuinka paljon seuraava ylempi oktaavi vaikuttaa lopputulokseen: kasvattamalla amplitudia saadaan terävämpiä yksityiskohtia ja samalla rosoisempaa lopputulosta.



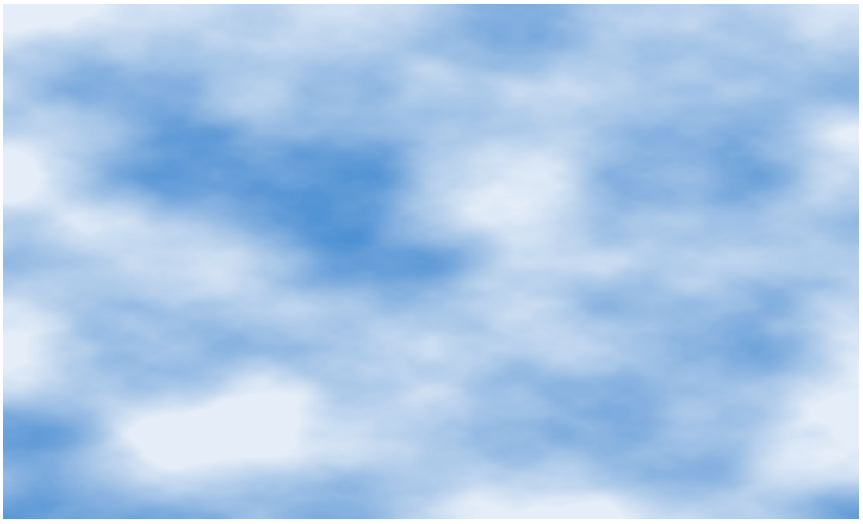
Kuva 1. Kahden, neljän ja kahdeksan oktaavin kohinaa.

Perlin-kohinaa ei ole mitenkään vaikea toteuttaa omassa ohjelmassa: tarvitaan vain satunnaislukuja, niiden interpolointia tavalla tai toisella sekä eri oktaavien yhteenlasku. Processingissa, jolla esimerkkiohjelma on tehty, kaikki löytyy jo valmiina standardikirjastosta, joten yksi työvaihe voidaan jättää pois. Keskeisimmät tarvittavat funktiot ovat kohina-arvon annetussa kohdassa laskeva *noise* sekä oktaavien määrän ja amplitudin säätävä *noiseDetail*. Kolmas mahdollisesti hyödyllinen funktio on *noiseSeed*, jolla voi pakottaa arvot aina samoiksi – muussa tapauksessa joka ajokerralla saadaan erilainen lopputulos.

Kohinasta maastoksi

Esimerkkiohjelma luo Perlin-kohinaa käyttäen kuvan 2 mukaisen saaristomaiseman jylhine kalliorantoineen ja merenpohjineen. Piirto tapahtuu reaaliajassa, joten kameraa voi liikuttaa ympäriinsä. Tässä tosin tyydytään vain paikallaan pyörimiseen, sillä maailman kultakin reunalta voi helposti pudota tyhjyyteen kuin Flat Earth Societyn jäsen konsanaan.

Ohjelman alustusvaiheessa täytetään kaksiulotteinen *elevation*-taulukko sopivalla Perlin-kohinalla. *Noise*-funktio toimii kätevästi sekä yhdessä, kahdessa että kolmessa ulottuvuudessa, mikä säästää ohjelmointivaivaa. Oktaaveja



Kuva 3: Perlin-kohinalla luotuja pilviä.

on käytössä kuusi, mikä tuottaa vielä verrattain pehmeitä muotoja. Valaistusta varten kullekin kulmapisteelle täytyy laskea myös normaalivektori, joka tässä tapauksessa muodostetaan keskiarvona sivu- ja syvyyssuunnan naapuripisteistä.

Processingissa on valmiit, joskin melko rajoittuneet työkalut 3D-grafiikan piirtämiseen, joten perspektiivin ja koordinaattimuunnosten laskennasta ei tarvitse tässä tapauksessa huolehtia itse. Riittää, että ikkunan koon asettavalle *size*-funktiolle annetaan kolmantena parametrina *P3D*, jonka jälkeen Processing piirtää grafiikan Javan OpenGL-kirjaston avulla. Hieman kömpelönä piirteenä koordinaatisto on oletuksena vasenkätinen – matemaattinen koordinaatistohan on puolestaan oikeakätinen – ja pikseleihin perustuva, mutta kumpikin epäkohta on onneksi helposti korjattavissa.

Itse piirto on suoraviivaista: maisema piirretään suikale kerrallaan kolmionauhoina käyttäen y-koordinaattina *elevation*-taulukosta luettavaa korkeusarvoa. Toisin sanoen x/z-suunnassa käydään läpi ruudukko, jonka kunkin kulmapisteen korkeusarvo saadaan aivan alussa lasketusta Perlin-kohinasta. Kullekin kulmapisteelle täytyy antaa myös normaali, jotta *directionalLight*-funktiolla luotu aurinko valaisee rinteet oikein.

Viimeisenä askeleena piirretään vielä meri, joka tässä on tapauksessa vain yksi iso vaakasuuntainen neliö. Piirtovärissä on mukana alfa-kanava, joten veden alle jääneet pinnanmuodot kuultavat läpi. 3D-kirjastossa on vakiona päällä z-puskuri, joka huolehtii siitä, että vesi ei peitä alleen vuorten korkeita kohtia.

Katson sineen taivaan

3D-maasto on nyt saatu piirrettyä. Seuraavaksi voidaan miettiä, miten sitä voisi edelleen kohentaa. Esimerkkiohjelma on tarkoituksella yksinkertainen ja ymmärrettävä, mutta lopputulos jää tässä muodossaan vielä kauas pelien tai elokuvien hienoista maisemista.

Aution yksivärisen taivaan elävöittäminen on ensimmäisiä mieleen tulevia parannuskohteita. Aurinko, pilvet ja ilmakehän efektit lisäisivät kukin näkymän realismia. Perlin-kohina tulee jälleen apuun, koska sillä saa helposti aikaan pilviharsoa, jota voi käyttää taivaan teksturointiin. Kuvassa 3 näkyy ensimmäinen, viidessä minuutissa Processingilla tehty yritelmä, joka näyttää jo varsin kohtuulliselta.

Vedenpinta on sekin toistaiseksi kovin muovisen oloinen – vesi saisi mieluusti heijastaa vuoria ja taivasta. Mahdollisia toteutustapoja heijastukselle on erilaisia, mutta suoraviivaisinta on piirtää vedenpinnan yläpuolelle jäävä vuoriston osa puoliläpinäkyvänä ylösalaisin veden alle. Samalla täytyy peilata myös valot sekä normaalivektorit. Lattea vedenpinta elävöityisi vesitekstuurilla tai mieluummin sopivalla varjostimella (shader), jolla mukaan saisi myös auringon heijastukset.

Seuraavia askeleita

Tasaisen harmaata vuoristoa on sitäkin helppo elävöittää monin tavoin. Esimerkiksi valokuviin perustuva pintakuvio on ensimmäinen askel realistisuutta kohti. Litteyttä poistaisi niin ikään kuhmu-, syvyys- tai normaalikartta, jolla valo heijastuisi pinnasta vaihtelevammin. Esimerkissä on vakiona käytössä Gouraud-varjostus, joka ei pärjää hyvin etenkään terävien muotojen tai kiiltävien pintojen kanssa.

Näin yksinkertainen maasto jaksaa vielä pyöriä hyvin reaaliajassa nykykoneilla, mutta tarkkuutta ja kokoa lisätessä hidastuminen iskee nilkkaan varsin nopeasti. Teknisenä nopeutusratkaisuna kulmapisteet, normaalivektorit ym. kannattaisi säilyttää mahdollisimman pitkälle näytönohjaimen sisäisessä muistissa. Tässä käytetty suora yksittäisten kolmioiden piirtäminen tekee koodista hieman helppolukuisempaa, mutta on tarpeettoman hidasta, etenkin kun geometria ei muutu ajon aikana mihinkään.

Tällä hetkellä vuoristosta piirretään säntillisesti joka kolmio, vaikka ruudulla näkyy vain murto-osa maastosta. Merkittävä algoritminen parannus olisi rajoittaa piirto näkökartioon (view frustum). Tätä varten sopiva tietorakenne on esimerkiksi nelipuu, joka mahdollistaa suurten nelikulmaisten alueiden hylkäämisen kerralla: jos jokin neljännes tai hierarkian seuraava taso on kokonaan ulkona kartiosta, voidaan kaikki sen alla olevat kolmiot jättää suoraan piirtämättä.

Kaukaisuudessa ei tarvita samanlaista tarkkuutta kuin katsojan lähellä. Etäisimmät vuoret häivytetään usein sumuun syvyysvaikutelman tehostamiseksi ja toisaalta tarvittavan piirtoetäisyyden rajoittamiseksi. Toinen optimointikeino ovat tarkkuustasot (LOD): geometriasta säilytetään kevennettyjä versioita, joita piirretään suurilla etäisyyksillä, kun täydelle tarkkuudelle ei ole tarvetta. Nopeutuksen vastapainoksi tarkkuustasoista seuraa myös hieman ylimääräistä päänvaivaa, sillä tasojen välille ei saa jäädä näkyviä saumoja.

Processing-esimerkkiohjelman voi ladata Skrollin www-sivulta: http://skrolli.fi/numerot/. Syvällisemmin reaaliaikaisen vuoristomaiseman luomisesta on kertonut mm. Iñigo "iq" Quilez esitelmässään "Behind Elevated": http://www.iquilezles.org/www/material/function2009/function2009.pdf