Tampereen yliopisto

Jussi Iho

PROSEDURAALISET KOHINAGENERAATTORIT

Teoria, toteutus ja soveltaminen

Informaatioteknologia ja viestintä
Kandidaatintyö
Huhtikuu 2019

TIIVISTELMÄ

Jussi Iho: Proseduraaliset kohinageneraattorit
Kandidaatintyö
Tampereen yliopisto
Tietotekniikan tutkinto-ohjelma
Huhtikuu 2019

Tässä työssä käsitellään proseduraalisten kohinageneraattorien teoriaa, toteutusta ja sovellustapoja, minkä pyrkimyksenä on luoda kattava yleiskuva aihekokonaisuudesta. Erityisenä näkökulmana aiheeseen ovat tietokonegrafiikka ja muu proseduraalinen sisällön generointi ohjelmistoissa. Tästä huolimatta keskeiset löydökset ja periaatteet esitellään tavalla, joka on yleisesti hyödynnettävissä myös muissa sovelluksissa.

Työn alussa keskitytään löytämään määritelmä kohinafunktioille ja niihin liittyville käsitteille keskustellen samalla niiden ominaisuuksista sekä tila- että taajuustasossa. Lisäksi esitellään useita kohinan manipulointimenetelmiä, joiden avulla pystytään luomaan kokonaan uusia kohinatyyppejä ja mitä erilaisimpia kuvioita ja efektejä.

Yleisen teoreettisen tarkastelun jälkeen siirrytään keskustelemaan käytännöllisistä kohinaalgoritmeista, joista yksinkertaisimpana menetelmänä käydään läpi arvokohina. Sen jälkeen syvennytään Perlin-kohinafunktion toteutukseen. Tähän sisältyy muun muassa Perlin-kohinan historiallisia referenssitoteutuksia ja niiden pohjalta kehitetty vaihtoehtoinen Perlin-kohinan toteutus, jonka uusi hajautusfunktio korjaa suuren osan klassisen version puutteista.

Viimeiseksi uuden kohinafunktion toteutusta verrataan klassiseen niiden suorituskyvyn ja laadullisten tekijöiden osalta. Tämän jälkeen toteutettuja kohinafunktioita käytetään monipuolisilla tavoilla havainnollistamaan kohinan esimerkkisovelluksia tietokonegrafiikassa ja maaston generoinnissa yhdistäen edeltävissä osioissa esille nousseita tekniikoita ja löydöksiä.

Avainsanat: kohina, proseduraalisuus, tietokonegrafiikka, tekstuurit, maastogenerointi

SISÄLLYSLUETTELO

1 JOHDANTO

Monia sekä luonnollisia, että ihmisen tuottamia stokastisten prosessien muokkaamia ilmiöitä ja niissä muodostuvia kuvioita tarkasteltaessa voidaan huomata, että niitä voidaan mallintaa ajan tai paikan funktiona muuttuvana kohinana. Erityisenä havaintona niin kutsutut $1/f$- ja $1/f^2$-taajuusjakaumat ovat yleisiä (West ja Shlesinger 1990; Ball 1999 s.211-215). Tähän liittyen luonnollisilla ilmiöillä on myös tunnetusti fraktaalisia ominaisuuksia (Mandelbrot 1983), jolloin puhutaan itsesimilaarisesta tai skaalainvariantista kohinasta. Esimerkkeinä tästä toimivat maan pinnan muoto (Gagnon et al. 2006), luonnon tekstuurit (Tolhurst et al. 1992; Balboa ja Grzywacz 2003), orgaanisen kudoksen rakenne ja toiminta (Havlin et al. 1995), sääolosuhteet (Blender et al. 2011) ja lukemattomat muut ilmiöt.

Vastaavasti monissa tietoteknisissä sovelluksissa yritetään löytää tapoja jäljitellä luonnollisia ilmiöitä. Ei ole siis sattumaa, että kohinalla on paljon sovelluksia muun muassa tietokonegrafiikassa ja muun sisällön generoinnissa ohjelmistoissa, mikä muodostaa myös tämän työn näkökulman aiheeseen. Muutamana esimerkkinä tästä ovat proseduraaliset tekstuurit, geometria, efektit sekä animaatiot. Lisäksi kohinaa voidaan hyödyntää erilaisissa simulaatioissa etenkin silloin, kun jonkin datan täydellinen simuloiminen olisi haastavaa tai jopa redundanttia, koska tuloksen ennalta tiedetään olevan lopulta vain kohinaa.

Yleisesti voidaan erottaa kaksi kilpailevaa paradigmaa, joiden perusteella ohjelmiston käyttämä data muodostetaan. Perinteisesti esimerkiksi tekstuurit, 3D-mallit ja muu ympäristö on määritelty eksplisiittisten proseduurien kautta, jolloin dataa käsitellään valmiina kokonaisuuksina ja tuodaan ohjelmaan ulkoisina tiedostoina tai ohjelmakoodiin sisään kovakoodattuna (Ebert et al. 2002, s.12-14). Haasteita kohdataan kuitenkin silloin, kun ohjelman sisältöä halutaan muunnella tai sitä tarvitaan suuri määrä, jolloin suositaan implisiittistä lähestymistapaa. Tällöin dataa ei enää tarvitse määritellä ennalta, vaan se korvataan proseduurilla, jonka tehtävänä on vastata dynaamisiin kyselyihin esimerkiksi väri-informaatiosta. Dataa siis voidaan arvioida ajon aikana proseduraalisesti, mistä kohinageneraattorit ovat esimerkki (Ebert et al. 2002, s.12-14).

Kohinageneraattoreiden käytöllä on esitetyissä sovelluksissa monia etuja. Niiden avulla n-ulotteista kohinadataa pystytään generoimaan missä tahansa avaruuden osissa millä tahansa resoluutiolla ja ilman, että dataa tarvitsee välttämättä säilyttää muistissa. Tietokonegrafiikassa tämä tarkoittaa, että korkeatarkkuuksiset tekstuurit, joiden koko lasketaan vähintään megatavujen luokassa, voidaan korvata muutamaa kilotavua vastaavalla määrällä koodia (Lagae et al. 2010). Sama pätee myös esimerkiksi proseduraalisesti generoituihin ympäristöihin, joista voidaan tehdä kooltaan käytännöllisesti äärettömän suuria. Kiinnostavaa on myös, että kohinan koostumusta on mahdollista säädellä parametrisesti, jolloin pystytään luomaan dynaamisia siirtymiä eri efektien välillä.

Tämän työn tavoitteena on luoda yleiskuva kohinageneraattorien teoriasta ja käytännön toteutuksesta, mihin sisältyy sekä matemaattisia, että ohjelmistoteknisiä näkökulmia ja reunaehtoja. Näihin kuuluvat ennen kaikkea kohinan suorituskyky ja laatu. Lisäksi työssä demonstroidaan joitakin kohinafunktioiden manipulointimenetelmiä ja käyttökohteita, vaikka niiden laajempi tarkastelu ei kuulukaan työn piiriin.

Erityiseksi tarkastelun kohteeksi valittiin Perlin-kohinafunktio, jota voidaan pitää yhtenä parhaiten tunnetuista kohina-algoritmeista. Perlin-kohinafunktion toteutukseen liittyen työssä esitetään myös yksi sen käyttämään hajautusfunktioon liittyvä parannusehdotus, joka korjaa joitakin aiempien toteutuksien puutteita. Lisäksi Perlin-kohinan ohella keskustellaan lyhyesti myös arvokohinasta, joka on edellä mainittua funktiota yksinkertaisempi algoritmi, mutta joka luokitellaan monin tavoin samaan kohina-algoritmien luokkaan Perlin-kohinan kanssa.

Työ jakautuu kohinan teoriaan, toteutukseen sekä tulosten arviointiin ja soveltamiseen. Luvussa 2 keskustellaan kohinafunktioiden yleisestä teoriasta sisältäen niiden matemaattista kuvailua, käsitteistöä ja keskeisiä konsepteja kohinan tuottamiseen ja manipulointiin liittyen. Tämän jälkeen luvussa 3 siirrytään käsittelemään varsinaisia käytännön kohina-algoritmeja esitellen ehdotetun Perlin-kohinafunktion toteutuksen lisäksi myös sen aiempia variaatioita. Lopulta luvussa 4 siirrytään analysoimaan toteutetun kohinafunktion suorituskykyä ja laatua, sekä esitellään sen muutamia esimerkkisovelluksia yhdistäen toisen ja kolmannen luvun löydöksiä.

2 KOHINAFUNKTIOIDEN YLEISKUVA

Käsitteellä *kohina* viitataan monien teknillisten aihepiirien tapauksessa usein erilaisiin satunnaisiin häiriöihin hyötysignaaleissa tai muussa analyysin kohteena olevassa datassa, jolloin pyrkimyksenä on suodattaa kohinaa pois tai vaihtoehtoisesti yrittää minimoida järjestelmän herkkyys sille. Tämän työn piirissä kohina kuitenkin määritellään laajemmin kaikkena jollain tavalla satunnaisena tai *näennäissatunnaisena* datana, joista jälkimmäinen termi liittyy erityisesti ohjelmistotekniikan näkökulmaan.

Vaikka kohinalle on kirjallisuudessa esitetty myös vielä spesifimpejä määritelmiä (esim. Lagae et al. 2010), kaikkien sovellusalojen näkökulmasta tyydyttävän määritelmän löytäminen on edelleen haastavaa. Samoin vaikeaa on täsmällisen rajan vetäminen proseduraalisen ja epäproseduraalisen ohjelmoinnin välille (Ebert et al. 2002, s.12).

Tästä huolimatta kohinan satunnaisuus on kaikkia sen määritelmiä yhdistävä tekijä. Tämän lisäksi kohinalla on suuri määrä muita matemaattisia ja ohjelmistoteknisiä ominaisuuksia, joita saatetaan haluta tavoitella tai välttää sovelluskohteesta riippuen parhaimman lopputuloksen aikaan saamiseksi.

2.1 Kohinafunktion määrittely

Pohjimmiltaan kohinadataa tuottavia kohinafunktioita voidaan pitää satunnaislukugeneraattoreina. Näillä funktioilla on siis joitakin satunnaislukugeneraattorien ominaisuuksia, kuten tietyn muotoinen arvojakauma, joka monien algoritmien kohdalla muistuttaa gaussista jakaumaa (Lagae et al. 2010). Tässä työssä käsiteltävien kohinafunktioiden tapauksessa tämä jakauma muodostaa rajatun arvovälin, kuten $[0, 1]$ tai $[-1, 1]$, mikä on hyödyllinen ominaisuus monien käytännön sovellusten kannalta.

Sen sijaan tavallisista satunnaislukugeneraattoreista poiketen useimmat kohinafunktiot ovat paikkasidonnaisia, mikä tarkoittaa, että kohinafunktion arvo riippuu aina avaruuden pisteestä $\mathbf{p}$, jossa funktiota arvioidaan (Ebert et al. 2002, s.67-68). Avaruuden, jossa piste sijaitsee, ulottuvuuksia ei ole rajoitettu, jolloin kohinadatasta muodostuu n-ulotteisia matriiseja.

Yleisesti voidaan esittää paikkasidonnaiselle kohinafunktiolle matemaattinen muoto

$$Noise_D(\mathbf{p}) \in \mathbb{R}, \quad \mathbf{p} \in \mathbb{R}^D,$$

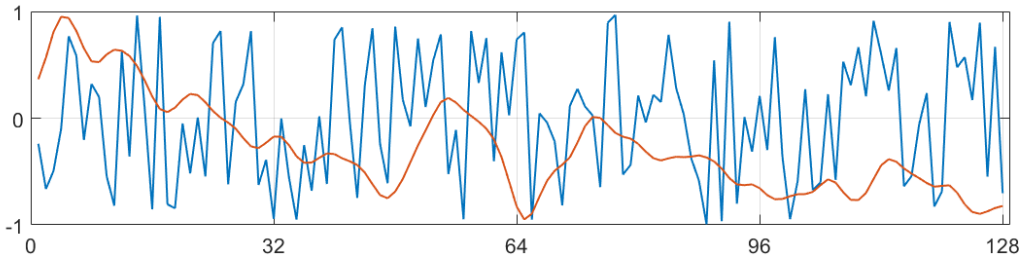
missä D on dimensioiden lukumäärä. Tämän työn piirissä kohinafunktiot ovat enimmäkseen 1-, 2- tai 3-ulotteisia, minkä takia saatetaan käyttää yksinkertaistettuja merkintöjä

$$Noise_1(\mathbf{p_1}) = Noise([x]) = Noise(x),$$
$$Noise_2(\mathbf{p_2}) = Noise([x,y]) = Noise(x,y),$$
$$Noise_3(\mathbf{p_3}) = Noise([x,y,z]) = Noise(x,y,z),$$

jolloin dimensioiden määrään viitataan koordinaattisyötteiden x, y, z lukumäärän kautta. Käytännön toteutuksissa kohinafunktioilla voi olla myös muita syötteitä, kuten siemenluku funktion arvojen satunnaistamista varten jokaisella ajokerralla.

Yksinkertaisimpana esimerkkinä kohinasta voidaan pitää niin kutsuttua *valkoista kohinaa*, jota myös eri ohjelmointikielten standardikirjastojen sisältämät tasaisen jakauman satunnaislukugeneraattorit tuottavat. Ideaalisesti tällaisten funktioiden tuottamat yksittäiset arvot eivät korreloi muiden arvojen kanssa millään tavalla. Valkoista kohinaa voidaan siis ajatella täydellisenä satunnaisuutena. (Ebert et al. 2002, s.67-68).

Satunnaislukuluonteensa mukaisesti myös kaikkien muiden kohinafunktioiden tuottamien arvojen ennustaminen yksittäisissä mielivaltaisissa sijainneissa on mahdotonta, vaikka – kuten myöhemmin huomaamme – kaikki kohina-algoritmit eivät toteutakaan tätä ominaisuutta täydellisesti. Erona kuitenkin on, että kohina-arvot voivat korreloida keskenään, mikä mahdollistaa kuvioiden muodostumisen kohinadataan. Tällöin kun arvo yksittäisessä pisteessä tunnetaan, voidaan suhteellisen suurella tarkkuudella ennustaa kohina-arvo myös sitä lähellä olevissa pisteissä ja useimmiten pystytään suoraan toteamaan, että kohinafunktio on jatkuva koko reaaliavaruudessa. Tällaiseen kohinaan saatetaan viitata *(spatiaalisesti) korreloivana*, *värillisenä* tai *koherenttina kohinana*. Esimerkkejä koherentista ja epäkoherentista kohinasta on esitetty kuvassa 2.1.



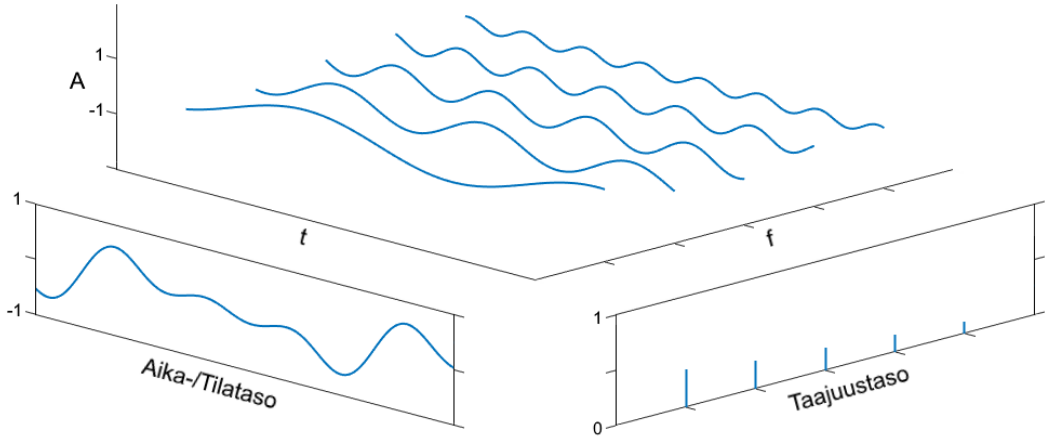
Kuva 2.1. *Esimerkkejä epäkoherentista (sininen) ja koherentista kohinasta (punainen). Molemmat kohinafunktiot ovat jaksottomia, paikkasidonnaisia ja näennäissatunnaisia.*

Muita perusmuotoiselle kohinafunktiolle tavoiteltavia matemaattisia ominaisuuksia ovat *stationaarisuus* ja *isotrooppisuus*, jotka tarkoittavat, että kohinafunktio säilyttää tilastolliset ominaisuutensa riippumatta siirrosta koordinaattijärjestelmässä tai koordinaatiston rotaatiosta (Perlin 1985a; Lagae et al. 2010). Jälleen tulemme myöhemmin huomaamaan, että kaikki algoritmit eivät täytä näitä ominaisuuksia täydellisesti.

2.2 Fraktaalinen kohina

Niin kutsutun aika- tai tilatason esityksen lisäksi kohinafunktioiden ominaisuuksia voidaan tutkia tarkastelemalla niitä taajuustasossa. Tällöin funktio voidaan tunnetusti esittää sinusoidisista funktioista koostuvana Fourier-sarjana, jonka taajuus- ja amplitudisisältöä pystytään analysoimaan. Havainnekuva tilanteesta on esitetty kuvassa 2.2.

Valkoisen kohinan taajuusjakauma on tunnetusti tasainen, eli se sisältää kaikkia taajuuskomponentteja niin, että niiden amplitudi on vakio ja taajuudesta riippumaton. Sen sijaan koherentti kohina sisältää enemmän suhteellisen matalia taajuuksia, minkä seurauksena tilatason signaali tasoittuu.



Kuva 2.2. *Saman yksiulotteisen funktion esitys aika- tai tilatasossa, sekä taajuustasossa. Huomautettakoon, että ero aikataso- ja tilataso-termien käytön välillä on tässä tapauksessa tulkinnanvarainen.*

Käytännössä kohinafunktioiden tuottama taajuuskaista on rajoittunut tietylle niille ominaiselle välille, mikä tuottaa tietyn piirrekoon kohinadatassa (Lagae et al. 2010). Tästä syystä analogisesti jaksollisten funktioiden kanssa perusmuotoista kohinafunktiota on mahdollista skaalata hallitusti amplitudin ja taajuuden funktiona

$$sNoise(\mathbf{p}) = A \cdot Noise(f \cdot \mathbf{p}), \tag{2.1}$$

jolloin arvoalueeksi saadaan tyypillisesti $[0, A]$ tai $[-A, A]$ ja funktiolle ominainen taajuuskaista muuttuu f-kertaiseksi. Taajuuden muutos voidaan ilmaista myös niin, että kohinafunktion tuottamien kuvioiden kokoluokka eli aallonpituus muuttuu vastaavasti $1/f$-kertaiseksi (Ebert et al. 2002, s.585). Näitä ominaisuuksia hyödyksi käyttäen voidaan muodostaa uudenlainen kohinafunktio, joka on eritaajuisten ja amplitudisten kohinafunktioiden summa

$$fNoise(\mathbf{p}) = \sum_{n=0}^{N_f-1} A_n \cdot Noise(f_n \cdot \mathbf{p}). \tag{2.2}$$

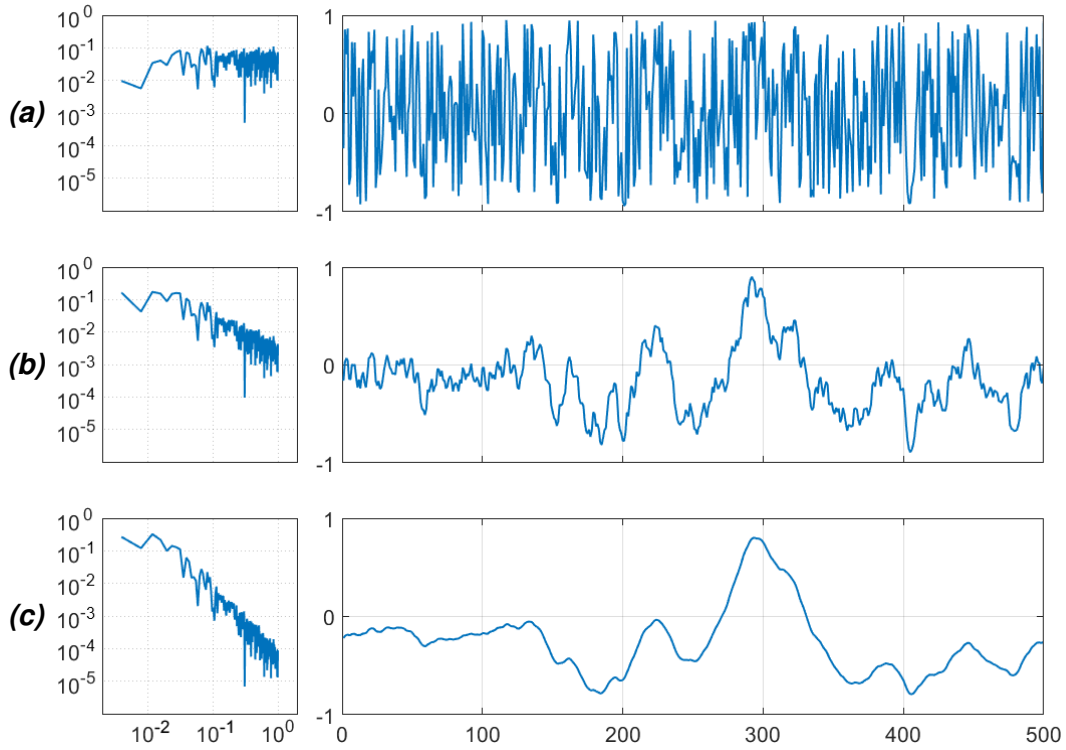
Termit f_n ja A_n voidaan määritellä eri tavoin n-arvon funktioina, jolloin voidaan hallita summafunktion kokonaistaajuusjakaumaa. Näistä amplitudi usein valitaan esittää taajuuden funktiona

$$A_n = \frac{1}{f_n^{\alpha}}, \tag{2.3}$$

missä α on vakio, joka määrää usean taajuuskomponentin läsnä ollessa koko signaalin taajuusjakauman muodon.

Odotetusti valkoisen kohinan tapauksessa $\alpha = 0$, jolloin taajuusjakauma on tasainen, kun taas koherentille kohinalle vakio on positiivinen. Kohinaa, joka noudattaa tätä taajuusjakaumaa, jossa taajuuskomponentin amplitudi on kääntäen verrannollinen taajuuteen tai sen johonkin potenssiin, kutsutaan $1/f^{\alpha}$-kohinaksi. Sen erityistapauksia ovat *vaaleanpunainen kohina* ($\alpha = 1$) ja *punainen kohina* ($\alpha = 2$), joista jälkimmäiseen viitataan usein myös *Brownin liikkeenä*. (Ball 1999, s.212)

Kuvassa 2.3 on havainnellistettu eksponentin vaikutusta tilatason signaaliin suodattamalla valkoista kohinaa taajuustasossa.



Kuva 2.3. *(a) Valkoista kohinaa ja siitä suodattamalla saadut (b) vaaleanpunaista $1/f$- ja (c) punaista $1/f^2$-kohinaa sisältävät signaalit, sekä niiden taajuussisältö diskreettinä Fourier-muunnoksena (Huom. taajuuden normalisointi ja logaritminen asteikko)*

Myös taajuusjakauman hallintaan esitellään uusi parametri l, eli *lakunariteetti*, joka kuvaa taajuuksien välistä suuruussuhdetta (Mandelbrot 1983, ks. Ebert et al. 2002 s.585-586). Yhdessä sen kanssa taajuus esitetään usein n-termin funktiona

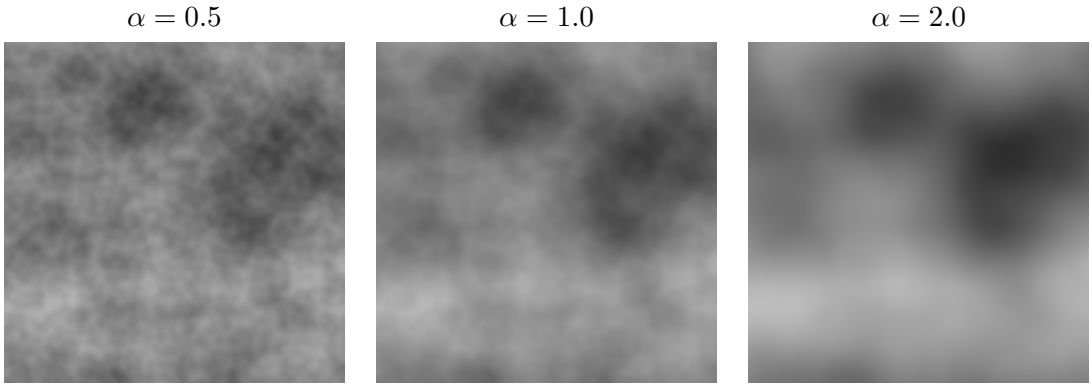
$$f_n = f_0 \cdot l^n, \tag{2.4}$$

missä f_0 on valinnainen parametri, joka määrittää sarjan alkutaajuuden ja voidaan valita sovelluskohteen tarpeiden mukaan. Yleisesti käytäntönä on että lakunariteetti on suurempi kuin 1.0, jolloin f_0-taajuudesta tulee tällöin summafunktion pienin taajuuskomponentti. Kokemusten mukaan moniin käyttötarkoituksiin hyvin soveltuva arvo on noin $l = 2.0$ (Ebert et al. 2002, s.585-586).

Yhdistämällä yhtälöt 2.3 ja 2.4 lausekkeeseen 2.2 saadaan lopputulokseksi α-, f_0-, l- ja N_f-parametrien kautta määritelty erityinen paikkasidonnainen $1/f^\alpha$-kohinafunktio, jolla on hyvin kontrolloitu taajuusjakauma

$$fNoise(\mathbf{p}) = \sum_{n=0}^{N_f-1} \frac{Noise(f_0 \cdot l^n \cdot \mathbf{p})}{(f_0 \cdot l^n)^\alpha}. \tag{2.5}$$

Esitetyn summafunktion eksponentti vaikuttaa kohinan koostumukseen samalla tavalla kuin kuvan 2.3 suodattamalla saaduissa esimerkeissä. Tämä voidaan havaita kuvassa 2.4 esitetyissä kaksiulotteisissa esimerkkikohinakuvissa.



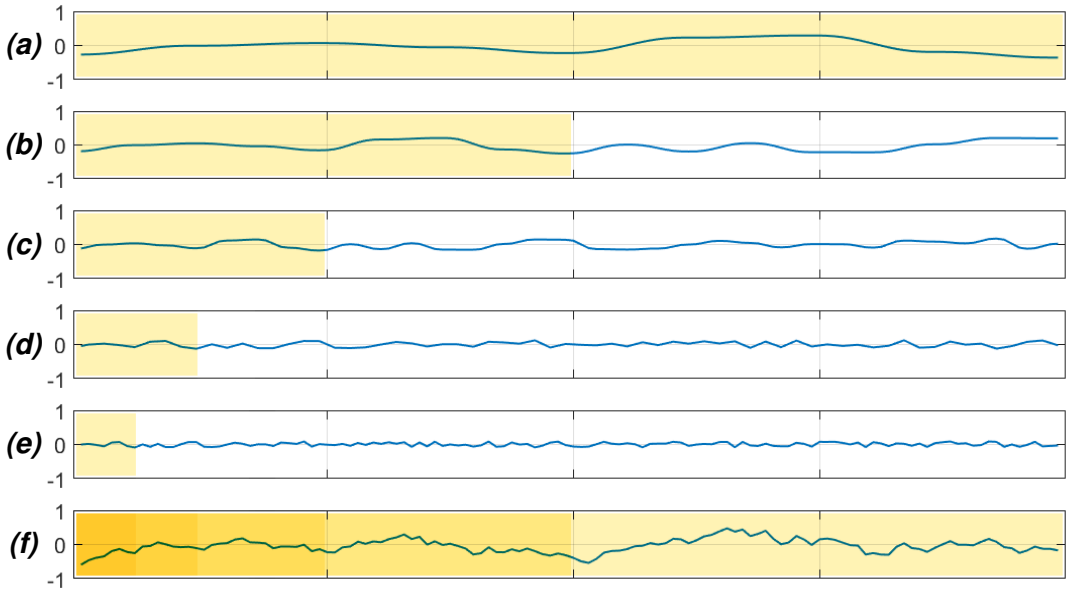
Kuva 2.4. *Eri α-arvoja vastaavia yhtälön 2.5 mukaisia kaksiulotteisia fraktaalisia kohinafunktioita ($l = 2.0$, $N_f = 5$)*

Ongelmaksi muodostuu tämän jälkeen vain hallita summafunktion arvoaluetta, joka on sen komponenttien arvoalueiden summa. Mikäli arvoväli halutaan normalisoida, on lopputulos jaettava vielä A_n-termien summalla. Tästä seuraa, että summa-arvo on usean satunnaismuuttujan painotettu keskiarvo, eli N_f-parametrin ollessa suuri ääriarvojen saavuttamisen todennäköisyys pienenee ja käytännössä saatava arvoalue kapenee, vaikka se teoreettisesti olisikin normalisoitu.

Toisena vaihtoehtona arvoalueen normalisointiin olisi generoida tarvittava määrä kohinadataa kerralla ja tämän jälkeen normalisoida sen arvoalue välille $[0, 1]$ maksimi- ja minimiarvojen mukaan. Tämä lähestymistapa voi toimia erittäin hyvin tietyissä sovelluksissa, joissa kohinadata tarvitsee generoida vain kerran, mutta yleiseksi ratkaisuksi ongelmaan se ei sovellu.

Summafunktion muodostaviin komponentteihin viitataan usein *oktaaveina* (Perlin 1985a), vaikka vastineestaan musiikissa poiketen taajuuden ei välttämättä tarvitse kaksinkertaistua jokaisella oktaavilla, koska suhde on riippuvainen lakunariteetista.

Eräs summafunktion mielenkiintoisimmista ominaisuuksista on sen tuottaman kohinan fraktaaliset piirteet, sillä skaalaamalla kohinaa ja summaamalla sitä itseensä muodostuu pienempiä ja pienempiä yksityiskohtia jotka sisältävät kopioita itsestänsä. Teoreettisessa tapauksessa, jossa N on ääretön, kohinafunktiolla olisi myös ääretön määrä yksityiskohtia. Tällaista kohinaa sanotaan matemaattisessa mielessä *itsesimilaariseksi* tai *skaalainvariantiksi*, kun taas kohinageneraattoreiden tapauksessa puhutaan *fraktaalisesta kohinasta*, josta on esitetty esimerkki kuvassa 2.5.



Kuva 2.5. *(a-e) Viiden kohinaoktaavin (f) summana generoitua fraktaalista kohinaa, josta nähdään että oktaavit ovat vain skaalattuja kopioita yhdestä ja samasta kohinafunktiosta* ($\alpha = 0.5$, $l = 2.0$)*.*

On myös syytä huomauttaa, että amplitudin ja taajuusjakauman ei ole pakollista seurata kaavojen 2.3 ja 2.4 mukaisia riippuvuuksia. Etenkin käytännön sovelluksissa ne voidaan määrittää millä tahansa halutulla tavalla johtaen erilaisiin tilatason tuloksiin (Cook ja DeRose 2005).

2.3 Kohinan manipulointi

Perusmuotoiset kaistarajoitetut kohinafunktiot, kuten luvussa 3 keskusteltavat arvo- ja Perlin-kohina, tuottavat koostumukseltaan vain tietynlaisia kohinakuvioita. Vaikka tämä onkin riittävää joidenkin sovellusten tarpeisiin, kohinageneraattoreita on mahdollista hyödyntää paljon monipuolisemmin erilaisten manipulointimenetelmien kautta. Esimerkkejä tästä ovat erityyppiset maskit, suodatukset, vääristymät ja parametrien muuntelutavat. Näin voidaan tuottaa paljon suurempi kirjo proseduraalisia kuvioita, efektejä ja animaatioita tai moniulotteista geometriaa.

Yhteistä näillä keskusteltavilla menetelmillä on, että ne eivät ole osa varsinaista kohinafunktiota, vaan sen sijaan pyrkimyksenä on joko manipuloida kohinafunktion parametreja, tai käsitellä kohina-arvoja jälkiprosessoinnissa. Käytännössä kyse on pitkälti luovasta ja kokeilevasta työstä, jossa ei välttämättä ole tiettyjä oikeita menetelmiä vaan suuri joukko vaihtoehtoisia tapoja, joiden kautta haluttuihin lopputuloksiin voidaan päästä. Tässä luvussa esitellään joitakin lähestymistapoja, mutta aiheen laajempi käsittely ei kuulu tämän keskustelun piiriin.

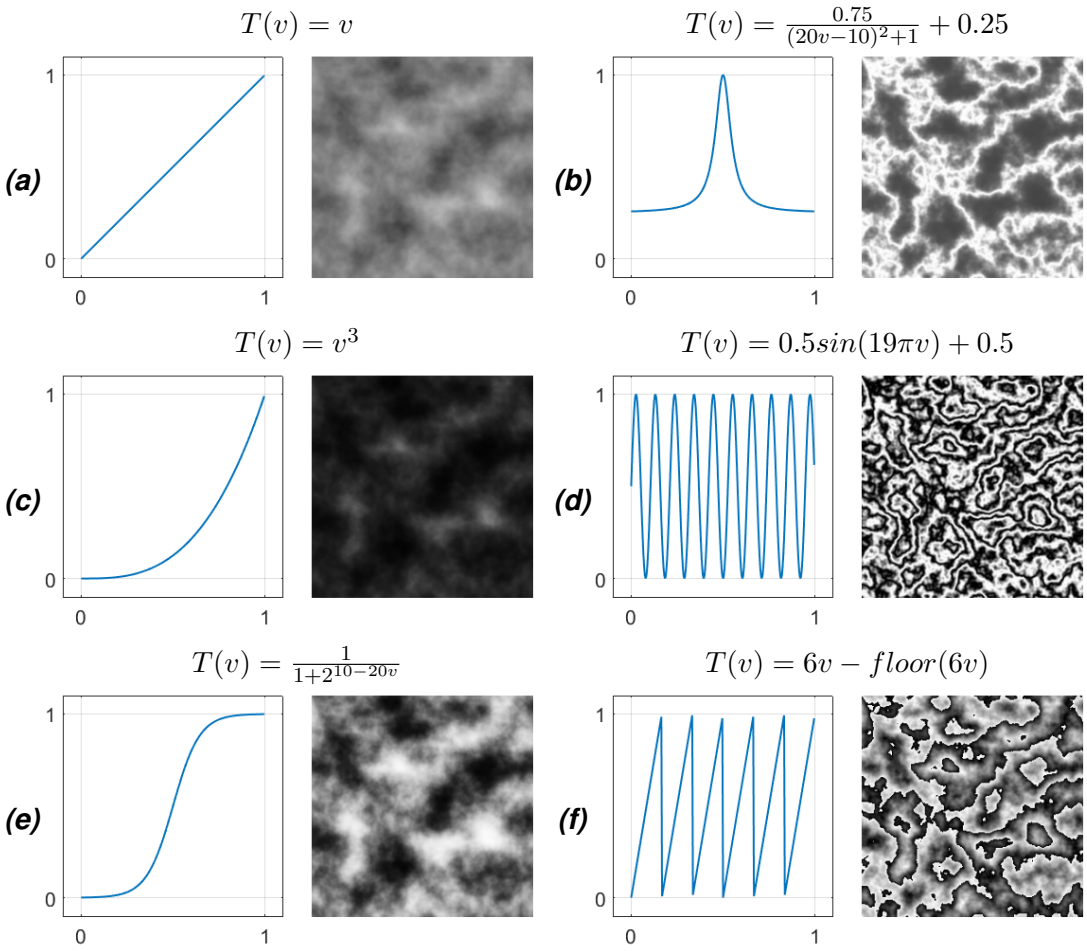
2.3.1 Pisteoperaatiot

Fraktaalisen kohinafunktion taajuusjakauma on helposti hallittavissa, mutta sen sijaan sen arvojakauma ei ole, vaan se on aina riippuvainen käytetystä perusfunktiosta. Jos arvojen jakaumaa haluttaisiin muokata, perusfunktiota ei kuitenkaan ole tarpeen vaihtaa, vaan jakaumaa voidaan muokata pisteittäin kohinafunktion ulkopuolella, eli jälkiprosessointina (Ebert et al. 2002, s.36-37).

Taajuustasossa tapahtuvalle tai muulle taajuuksien pois suodatukseen tähtäävälle ali- tai ylipäästösuodatukselle ei sen sijaan pitäisi olla yleensä tarvetta. Järkevämpää olisi tällöin jättää ei-halutut taajuuskaistat generoimatta alusta alkaen kohinageneraattorin parametreja säätelemällä. Kuitenkin on huomioitava, että arvoalueen muuntelu voi johtaa muutoksiin myös taajuusjakaumassa muilla tavoilla (Ebert et al. 2002, s.86).

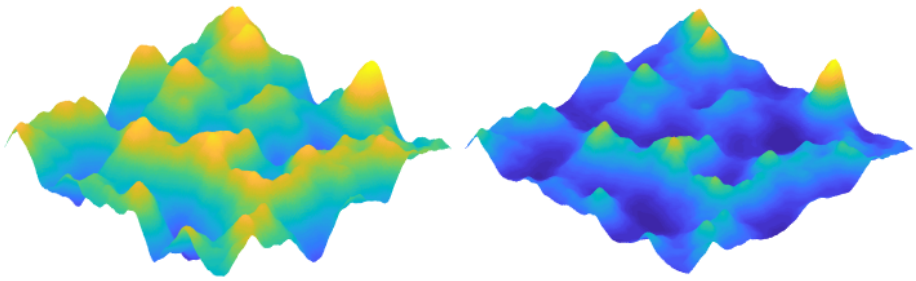
Tässä työssä *pisteoperaatioilla* viitataan sellaisiin matemaattisiin operaatioihin, jotka voidaan suorittaa yksittäisissä pisteissä itsenäisesti ympärillä olevista arvoista. Tällaisten operaatioiden etuna on että ne ovat hyvin yksinkertaisia ja suoraviivaisia.

Yleisesti pisteoperaatiot pystytään määrittelemään muunnosfunktiona $T(v) \in [0, 1]$ tai $[-1, 1]$, missä v on yksittäinen kohina-arvo jossain avaruuden pisteessä. Muunnosfunktiona voidaan käyttää mitä tahansa matemaattista funktiota, jolla saavutetaan haluttu tulos, eli se voisi olla myös paloittain määritelty. Lisäksi sen suunnittelussa voidaan hyödyntää tietoa perusfunktion arvojakaumasta. Esimerkkejä erilaisista funktioista ja niitä vastaavista kohinaefekteistä on esitetty kuvassa 2.6.



Kuva 2.6. *Useita pisteoperaatioina toteutettavia yksikanavaisia arvomuunnosfunktioita ja niitä vastaavia kaksiulotteiseen kohinadataan kohdistettuja efektejä.*

Pisteoperaatioiden kautta kohinadataan voidaan luoda erilaisia kuvioita, kuten saarekkeita tai renkaita. Vaihtoehtoisesti kohinan kontrastia tai kirkkaustasoa pystytään muokkaamaan. Kirkkauden säätelystä esimerkkinä on kuvassa 2.6c esitetty kuvankäsittelystä tunnettu gammakorjausfunktio $T_\gamma(v) = v^\gamma$ (Ebert et al. 2002, s.37-38), jonka vaikutusta kohinadatasta generoituun pintaan on havainnollistettu kuvassa 2.7.



(a) ***(b)***

Kuva 2.7. *(a) Muokkaamaton kohinadatasta generointu korkeuskartta ja (b) gammakorjauksen ($\gamma = 4$) vaikutus siihen.*

Joskus kohinadatasta saatetaan haluta generoida useita kanavia dataa eri muunnosfunktioiden kautta. Yksittäinen harmaasävyinen kohinakuva voidaan muuntaa esimerkiksi värilliseksi tekstuuriksi eli RGB-/RGBA-kuvaksi. Joitakin esimerkkejä tällaisista muunnoksista ja muista efekteistä on esitetty luvussa 4.2.

Kohinadatalle voidaan tehdä myös muun kaltaisia pisteittäisiä muunnoksia, kuten *maskioperaatioita*, joissa kohinafunktiota tai muuta matemaattista funktiota voidaan käyttää maskina muille kohinafunktioille, jolloin dataan muodostuu toisistaan erillisiä alueita. Samoin kohinan eri parametrejä voidaan säädellä samanlaiseen tapaan pisteittäin.

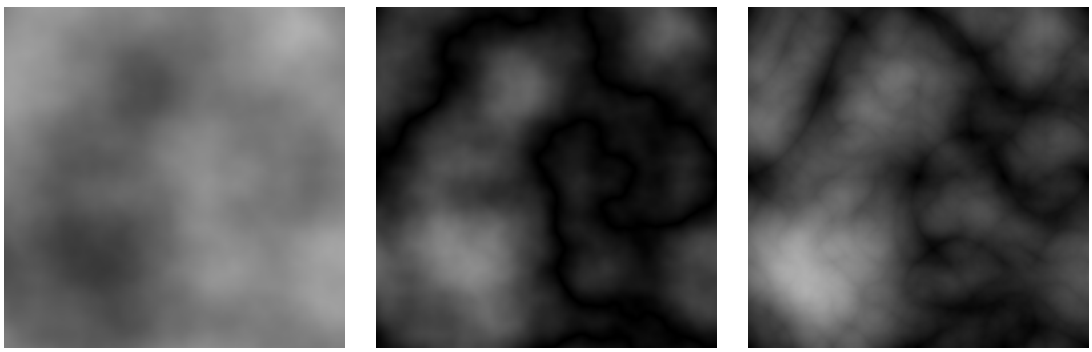
2.3.2 Turbulentti kohina

Fraktaalisessa kohinassa pisteoperaatiot eivät välttämättä ole vaihdannaisia oktaavien summan laskennan kanssa. Siinä missä edeltävässä alaluvussa 2.3.1 muunnosoperaatiot on kohdistettu koko fraktaalisummaan, ne voidaan myös suorittaa yksi oktaavi kerrallaan ennen summan muodostamista, minkä tuloksena voidaan synnyttää *turbulenttia kohinaa*. Tällöin aiempi yhtälö 2.2 saa uuden muodon

$$tNoise(\mathbf{p}) = \sum_{n=0}^{N_f-1} A_n \cdot T(Noise(f_n \cdot \mathbf{p})), \tag{2.6}$$

missä T on aiemmin esitelty pistemuunnosfunktio. Sen paikalla voidaan jälleen käyttää mitä tahansa matemaattista operaatiota, mutta luultavasti perinteisin turbulentin kohinan funktiotyyppi on itseisarvofunktiot (Ebert et al. 2002, s.85-87). Tästä huolimatta tässä työssä turbulentilla kohinalla viitataan kaikkiin yhtälön 2.6 mukaisiin kohinafunktioihin.

Odotetusti itseisarvofunktio voi saada vaihtelevia muotoja riippuen perusfunktion arvoalueesta ja -jakaumasta. Välin $[0, 1]$ mukaista arvoaluetta vastaava itseisarvofunktio voisi esimerkiksi olla muotoa $T(v) = abs(2v - 1)$. Kuvassa 2.8 nähdään kyseessä olevan funktion vaikutus fraktaalisumman ulko- ja sisäpuolella.



(a) ***(b)*** ***(c)***

Kuva 2.8. *(a) Tavallinen fraktaalinen kohinafunktio ($N_f = 5$, $\alpha = 1.5$, $l = 2.0$) sekä (b) siihen kohdistetun pisteittäinen arvomuunnoksen tulos ja (c) oktaaveittain tehdyn muunnoksen kautta syntynyt turbulentti kohinafunktio, missä $T(v) = abs(2v - 1)$.*

Huomataan että itseisarvofunktio tuottaa kohinafunktion arvoalueeseen terävän epäjatkuvuuskohdan, josta seuraa, että kohinaan muodostuu "kumpuileva"rakenne (Ebert et al. 2002, s.85-87). Mikäli funktion arvoalueen jatkuvuus olisi tärkeää, voitaisiin käyttää jotain muuta jatkuvaa itseisarvon approksimaatiota. Muunnosfunktio voitaisiin myös vaihtaa kokonaan toiseen operaatioon, jolloin kohinan rakenteesta tulisi täysin erilaista.

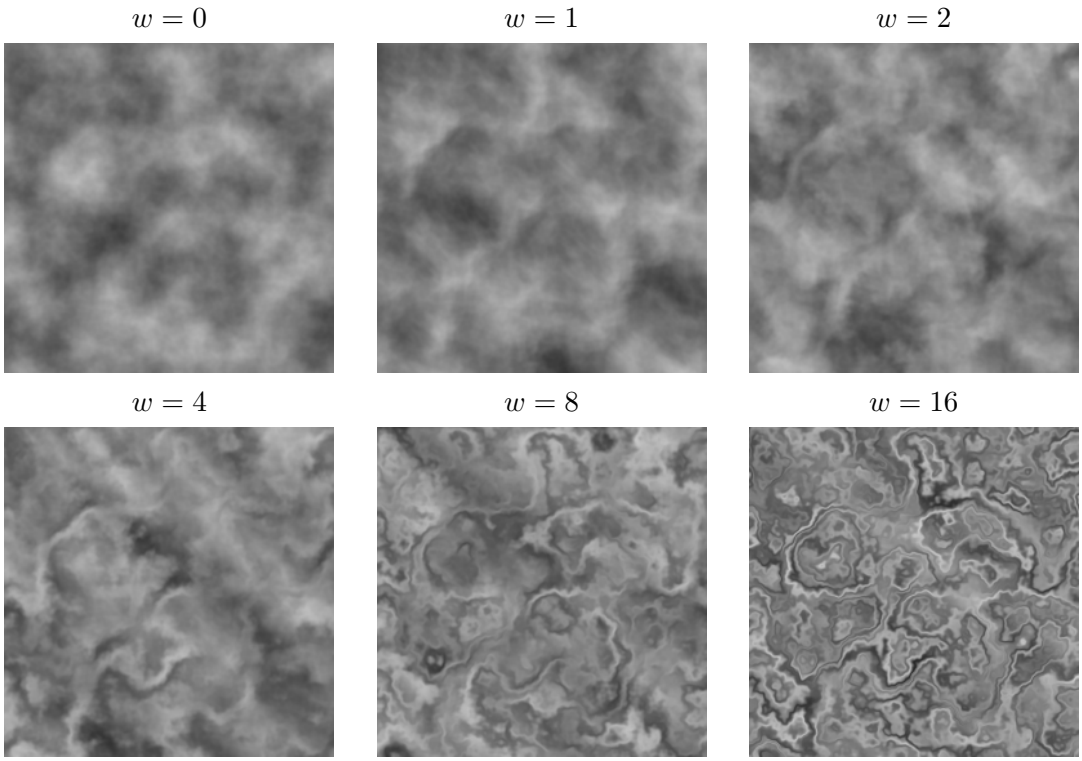
2.3.3 Siirtovääristymät

Toisena merkittävänä kohinan manipulointimenetelmänä voidaan esittää erilaiset vääristymät, joita voidaan tuottaa siirtämällä kohinafunktiota epätasaisesti tilatasossa. Tämän työn yhteydessä näihin menetelmiin viitataan *siirtovääristyminä*, mutta joissakin yhteyksissä ilmiöstä puhutaan myös *perturbaatioina* (Ebert et al. 2002, s.89-94).

Yleisesti kohinafunktion siirto voidaan toteuttaa jonkinlaisen matemaattisen vääristymäfunktion $\mathbf{Q}(\mathbf{p}) \in \mathbb{R}^D$ kautta, mikä voidaan esittää muodossa

$$pNoise(\mathbf{p}) = fNoise(\mathbf{p} + w \cdot \mathbf{Q}(\mathbf{p})), \tag{2.7}$$

missä valinnainen parametri w määrittää vääristymän voimakkuuden. Tyypillisesti vääristymäfunktiona käytetään jotain kohinafunktiota $\mathbf{Q}(\mathbf{p}) = Noise(\mathbf{p})$. Kohinafunktio voi jopa olla itse itsensä vääristymäfunktio. Kuvassa 2.9 on esitetty tällaisia yhtälön 2.7 mukaisia tuloksia eri w-arvoilla.



Kuva 2.9. *Kaksiulotteinen kohinafunktio, jota on siirretty avaruudessa oman arvonsa ja $w-$parametrin tulon verran molemmissa koordinaattisuunnissa.*

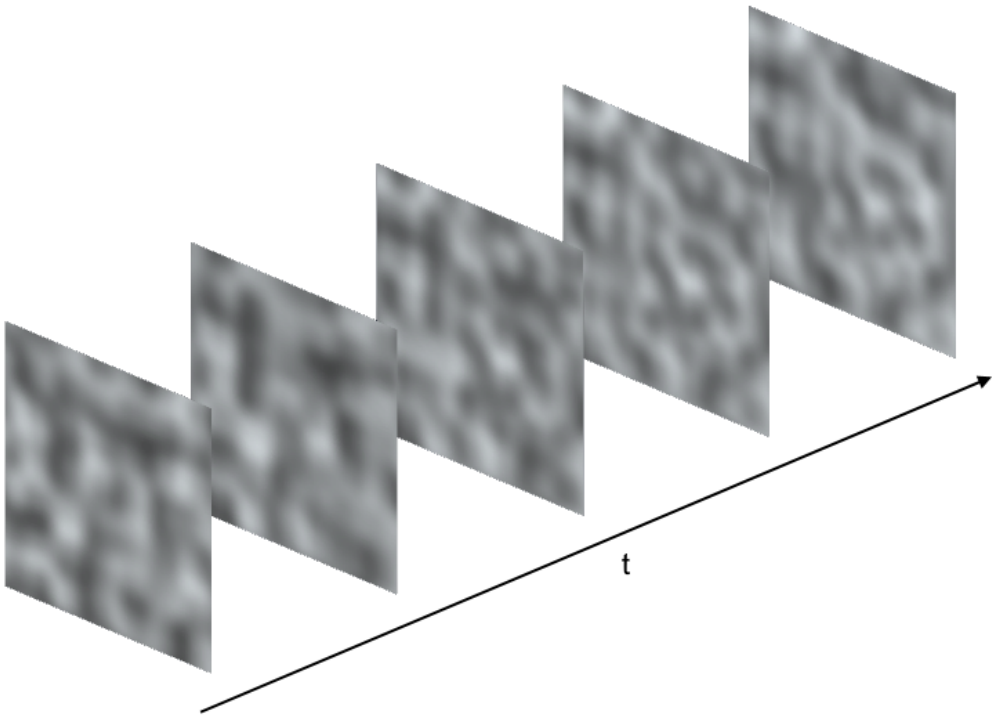
Menetelmän avulla pystytään luomaan hyvin suuri määrä uusia kohinakuvioita, joiden generointi pelkkien perusfunktioiden avulla olisi vähintäänkin ongelmallista. Siirtovääristymät ovat muun muassa luonnollinen vaihtoehto turbulenttien virtausten simulointiin.

Vääristymän kohteena voi olla myös jokin muu kuin kohinafunktio, kuten tekstuuri tai matemaattinen käyrä. Siirtovääristymiä hyödyntävistä kohinageneraattoreista on esitetty lisää esimerkkejä luvussa 4.2.

2.3.4 Animoitu kohina

Kohinaa voidaan käyttää hyödyksi myös reaaliaikaisissa animaatioissa, jolloin kohinafunktioon lisätään yksi ylimääräinen ulottuvuus, johon aikamuuttuja sijoitetaan. Esimerkki kaksiulotteisesta animoidusta kohinasta on esitetty kuvassa 2.10. Yleisesti animoitu kohinafunktio voidaan esittää muodossa

$$aNoise_D(\mathbf{p}, t) = fNoise_{D+1}([p_0, p_1, \ldots, p_{D-1}, t]). \tag{2.8}$$



Kuva 2.10. *Kaksiulotteinen animoitu kohinafunktio, jossa aika on kolmas muuttuja kolmiulotteisessa kohinafunktiossa.*

Toisin sanoen kolmiulotteisen kohinan animointi edellyttää jo neliulotteista kohinafunktiota ja kuten myöhemmin todetaan, dimensioiden määrän lisääminen voi aiheuttaa suorituskykyongelmia käytännön toteutuksissa. Kohinafunktioiden käytöllä animaatioissa on toisaalta joitakin etuja, koska ne ovat jaksottomia, dynaamisesti kontrolloitavissa ja tuottavat melko luonnollisia lopputuloksia.

3 KÄYTÄNNÖN KOHINA-ALGORITMIT

Luvussa 2.2 esitettiin, että koherenttia kohinaa on mahdollista generoida alipäästösuodattamalla valkoista kohinaa kuvan 2.3 tulosten mukaisesti. Käytännön ohjelmistoteknisissä sovelluksissa tämä ei ole yleensä käytännöllistä, koska se on laskennallisesti sekä raskasta, että vaikeaa toteuttaa staattisesti ja tilattomasti (Ebert et al. 2002, s.82-83).

Sen sijaan kohinan generointiin on olemassa suuri joukko erilaisia proseduraalisia algoritmeja, joiden tavoitteena on approksimoida ideaalista kaistarajoitettua valkoista kohinaa (Ebert et al. 2002, s.82-83). Näiden algoritmien toteutus sekä edellisessä luvussa esiteltyjen yleisten matemaattisten ominaisuuksien ja konseptien soveltaminen käytännössä tuovat mukanaan myös joukon uusia kysymyksiä.

Mitä tässä työssä asetettuihin reunaehtoihin tulee, yksi keskeisimpiä on kohinafunktion staattinen ja tilaton toteutus, mikä tekee monisäikeisestä laskennasta triviaalia ja mahdollistaa myös toteutuksen näytönohjaimessa melko suoraviivaisesti (esim. Perlin 2004; Olano 2005). Muita teknisiä vaatimuksia ovat reaaliaikaiseen laskentaan riittävä suorituskyky ja funktion satunnaistaminen siemenluvun avulla. Kohinafunktion on toteutettava myös luvuissa 2.1 ja 2.2 esitellyt keskeiset matemaattiset ominaisuudet, eli funktion on oltava määritelty koko reaaliavaruudessa, sillä on oltava rajallinen arvoväli ja taajuuskaista, sekä sen on oltava paikkasidonnainen ja jaksoton.

Kohina-algoritmien suorituskykyä olisi myös mahdollista optimoida luopumalla osasta esitettyjä reunaehtoja, mutta silloin löydökset eivät olisi enää yhtä yleiskäyttöisiä.

3.1 Yleisiä kohina-algoritmeja

Kohinan generointiin käytettyjä algoritmeja voidaan luokitella niiden erilaisten toimintaperiaatteiden ja muiden perusprimitiivien mukaan. Puhtaasti proseduraalisten kohinageneraattorien keskuudessa voidaan erottaa ainakin neljä tällaista luokkaa.

Tässä työssä käsitellään erityisesti *hilakohina-algoritmeja* (engl. *lattice noise*), joissa kohinan generointi perustuu avaruuden alijakoon jonkinlaiseksi ruudukoksi, jonka solujen sisällä kohinafunktion arvoa arvioidaan. Avaruuden alijako tällä tavalla yksinkertaistaa kohinafunktion toimintaa, mutta toisaalta se myös johtaa helposti enemmän tai vähemmän havaittaviin säännöllisyyksiin kohinakuvioissa. (Lagae et al. 2010)

Muita tunnistettavia luokkia, joita tässä työssä ei käsitellä, ovat muun muassa niin kutsutut *harvat konvoluutiokohinat* (engl. *sparse convolution noise*) (Lagae et al. 2010), *solukohinat* (engl. *cell noise*) (Ebert et al. 2002, s.135-136) ja erilaiset iteratiiviset prosessit, kuten MPD-pohjaiset algoritmit (engl. *midpoint displacement*) (Ramstedt ja Smed 2016).

3.1.1 Arvokohina

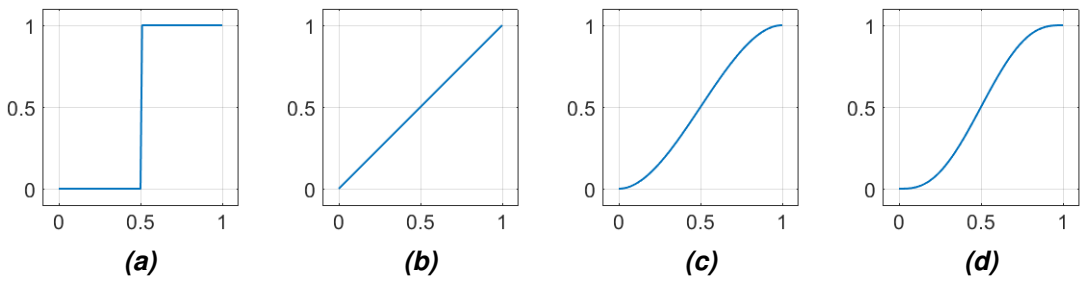
Koherenttien kohina-algoritmien keskuudessa niin kutsuttua *arvokohinaa* (engl. *value noise*) pidetään ehkä kaikkein yksinkertaisimpana ja nopeimpana vaihtoehtona. Koska reaaliaikaisten sovellusten keskuudessa on aina olemassa kysyntää laskennallisesti mahdollisimman tehokkaille ratkaisuille, arvokohina nousee edelleen esiin monissa artikkeleissa ja oppikirjoissa perusesimerkkinä kohina-algoritmeista. (esim. Ebert et al. 2002; Olano et al. 2002).

Arvokohina on hilakohina-algoritmi, jossa avaruus jaetaan yksikköväleihin, jotka voivat ulottuvuuksien määrästä riippuen olla neliöitä tai moniulotteisia kuutioita, ja joiden kulmakoordinaatit ovat kokonaislukuja. Pistettä ympäröivissä kokonaislukukoordinaateissa generoidaan tämän jälkeen jokaista kulmaa kohti yksi paikkasidonnainen satunnaisarvo, joka voi olla tasaisesti jakautunut tai noudattaa esimerkiksi gaussista jakaumaa. Tämän jälkeen yksikkövälin sisällä olevan pisteen arvo määritetään moniulotteisena interpolaationa kulma-arvojen välillä. (Ebert et al. 2002, s.70-72)

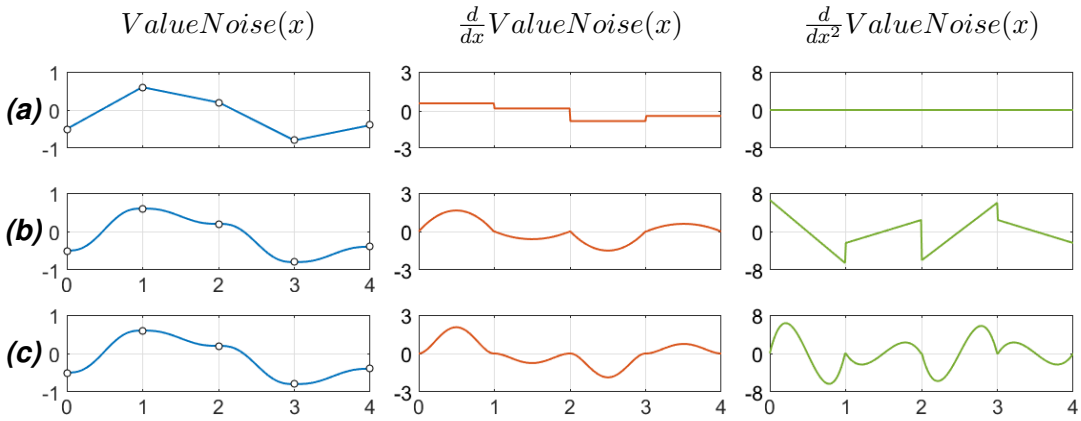
Sekä arvokohinassa, että esimerkiksi seuraavaksi esiteltävässä Perlin-kohinassa interpolointi toteutetaan *splini-interpolaationa*. Tällöin kahden tunnetun vierekkäisen pisteen välillä olevaa arvoa arvioidaan niiden välille sovitetun käyrän, eli splinin, avulla johtaen paloittain määriteltyyn funktioon. Käytettyjen splinipolynomien muodot vaihtelevat, mutta kuten tavallista, käytännössä kompromissi joudutaan tekemään laskentanopeuden ja tuloksen laadun välillä. Esimerkiksi usein käytetään ensimmäisen, kolmannen tai viidennen asteen polynomeja

$$\begin{aligned} s_1(t) &= t, \\ s_3(t) &= -2t^3 + 3t^2, \\ s_5(t) &= 6t^5 - 15t^4 + 10t^3, \end{aligned} \tag{3.1}$$

missä $t \in [0, 1]$. Keskeisenä syynä korkeamman asteen polynomien käyttöön on, että kolmannen ja viidennen asteen splinien avulla voidaan määritellä myös ensimmäinen tai toinen derivaatta jatkuvaksi, millä on arvokohinan ja monien muiden algoritmien tapauksessa suuri laadullinen vaikutus lopputulokseen (Perlin 2002b). Näiden polynomien ja niitä vastaavien kokonaisten arvokohinafunktioiden kuvaajat on esitetty kuvissa 3.1 ja 3.2.



Kuva 3.1. *(a) Lähimmän naapurin, (b) 1. asteen, (c) 3. asteen ja (d) 5. asteen interpolaatiosplinit.*



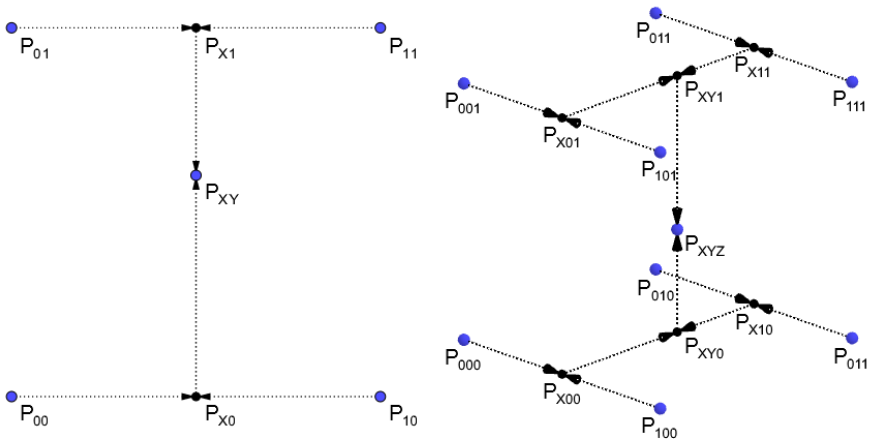
Kuva 3.2. *Yksiulotteinen arvokohinafunktio, jossa on käytetty (a) 1. asteen, (b) 3. asteen ja (c) 5. asteen interpolaatiosplinejä. Kasvattamalla polynomin astetta kohinafunktiolle saadaan jatkuva 1. ja 2. derivaatta, mitä voidaan pitää laadukkaampana lopputuloksena.*

Interpolaatiomenetelmän valinnan jälkeen tarvitaan ainoastaan jonkinlainen paikkasidonnainen satunnaislukugeneraattori $Rand(\mathbf{p}) \in [0,1]$, jonka toteuttamisesta keskustellaan enemmän luvussa 3.2. Näin ollen esimerkiksi 1- ja 2-ulotteisille arvokohinafunktioille voidaan esittää lausekkeet

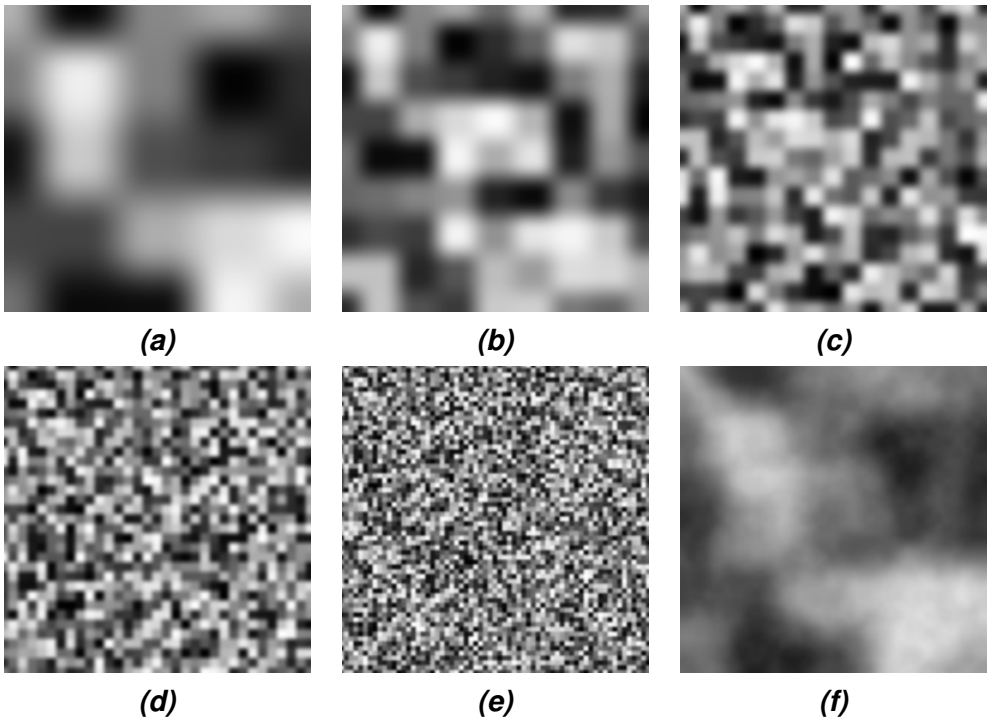
$$ValueNoise(x) = s(1-x_f) \cdot Rand(x_0) + s(x_f) \cdot Rand(x_1),$$

$$\begin{aligned} ValueNoise(x,y) = {} & s(1-y_f)(s(1-x_f) \cdot Rand(x_0,y_0) + s(x_f) \cdot Rand(x_1,y_0)) \\ & + s(y_f)(s(1-x_f) \cdot Rand(x_0,y_1) + s(x_f) \cdot Rand(x_1,y_1)) \end{aligned}$$

missä $[x_0,y_0] = floor([x,y])$, $[x_1,y_1] = ceil([x,y])$ ja $[x_f,y_f] = [x,y] - [x_0,y_0]$. Yleisemmin interpolaatio moniulotteisessa avaruudessa voidaan tehdä rekursiivisesti yhdessä koordinaattisuunnassa kerrallaan, minkä 2- ja 3-ulotteisista tapauksia on havainnollistettu kuvassa 3.3. Esimerkkejä kaksiulotteisista arvokohinaoktaaveista ja niiden fraktaalisummasta on esitetty kuvassa 3.4.



Kuva 3.3. *Havainnekuvat kaksi- ja kolmeulotteisesta interpoloinnista*



Kuva 3.4. *(a-e) Viisi oktaavia kaksiulotteista arvokohinaa ennen amplitudin muokkausta ja (f) niiden fraktaalisumma (*$\alpha = 1.5$*,* $l = 2.0$*) käyttäen 5. asteen interpolaatiota.*

Käytännössä arvokohina on siis skaalattua paikkasidonnaista valkoista kohinaa, jota generoidaan interpoloimalla reaaliavaruudessa diskreetin skalaarikentän sisällä. Kuvista näemme, että arvokohinan käyttämä avaruuden neliskanttinen alijako tuottaa hyvin näkyviä neliskanttisia artifakteja kun kohinaa generoidaan yksittäisinä oktaaveina, vaikkakin ne osittain kätkeytyvät fraktaalisen kohinan tapauksessa.

Yleisesti kohinafunktioiden laskennallinen kompleksisuus kasvaa ulottuvuuksien määrän mukana. Arvokohinan tapauksessa laskenta-aika on luokassa $O(2^D)$, koska huomataan, että paikkasidonnaisia satunnaisarvoja generoidaan yksi kappale yksikkövälin jokaista kulmaa kohti, eli yhteensä 2^D kappaletta. Tämän jälkeen yhden akselin suunnassa laskettavia interpolaatioita joudutaan laskemaan yhteensä $\sum 2^d$ kertaa, missä $d \in \{0, 1, 2, \ldots, D-1\}$. Muilta osin laskenta voidaan tehdä vakioajassa ainakin tässä työssä esitellyissä tilattomissa toteutuksissa.

Arvokohinan kanssa tehtävän laadullisen kompromissin myötä funktio toteuttaa vain osan luvuissa 2.1 ja 2.2 esitellyistä matemaattisista ominaisuuksista. Esimerkiksi tilatason kuvioiden neliskanttisuus ilmenee taajuustasossa niin, että taajuuskomponentit jakautuvat epätasaisesti akselien suuntaisesti, eli arvokohina ei ole isotrooppista ja se on vain heikosti kaistarajoitettua. Toisena suurena ongelmana on, että funktio tuottaa enemmistön ääriarvoistansa kokonaislukupisteissä johtuen siitä, että yksikkövälien keskellä arvot ovat usean satunnaismuuttujan keskiarvoja.

Näistä syistä arvokohinalle on olemassa myös vaihtoehtoisia algoritmeja, jotka pyrkivät orgaanisempiin lopputuloksiin suuremman laskenta-ajan hinnalla.

3.1.2 Perlin-kohina

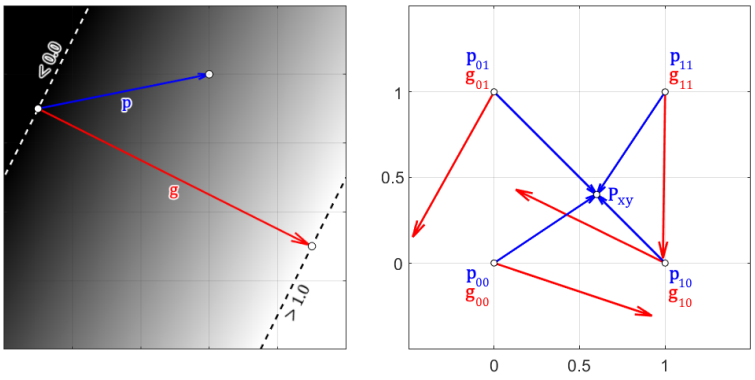
Yhtenä kuuluisimmista kohina-algoritmeista on alunperin Ken Perlinin (1985a) esittelemä hilapohjainen kohinafunktio, joka osittain korjaa monia arvokohinan ongelmia. Tämän lisäksi se edustaa laajempaa *gradienttikohina*-algoritmien luokkaa, missä kohinafunktion perusprimitiivinä toimii skalaarikentän sijaan vektorikenttä (Lagae et al. 2010).

Perlin-kohinassa avaruuden alijako on myös neliskanttinen ja kohina-arvo pisteessä määritellään täysin sitä vastaavan yksikkövälin sisällä. Tässä tapauksessa jokaista kulmaa vastaavaan kokonaislukupisteeseen arvotaan satunnainen gradienttivektori, jonka avulla lasketaan gradienttiarvo pisteen ympäristössä. Tämän jälkeen lopullinen arvo saadaan interpoloimalla gradienttiarvojen välillä n-ulotteisesti.

Väri- tai intensiteettigradientteja on mahdollista mallintaa hyvin yksinkertaisesti vektoriesityksen ja pistetulon avulla. Yleisessä tapauksessa yksittäisen värigradientin arvo mielivaltaisessa pisteessä voidaan esittää kuvan 3.5 mukaisesti paikkavektorin $\mathbf{p}$ ja gradienttivektorin $\mathbf{g}$ pistetulona

$$GradValue(\mathbf{p}, \mathbf{g}) = \frac{\mathbf{p} \cdot \mathbf{g}}{\mathbf{g} \cdot \mathbf{g}} = \frac{\mathbf{p} \cdot \mathbf{g}}{|\mathbf{g}|^2}. \tag{3.2}$$

Gradientin arvo kasvaa siis gradienttivektorin suuntaisesti ja on pienempi kuin 0.0 pisteen $\mathbf{p}$ ollessa gradienttivälin "takana"ja suurempi kuin 1.0 sen "edessä". Monissa toteutuksissa ja erityisesti 2-ulotteisen Perlin-kohinan tapauksessa gradienttivektoreiksi suositellaan yksikkövektoreita, jolloin nimittäjä $|\mathbf{g}|^2$ supistuu pois. Kuten luvussa 3.2 keskustellaan, on olemassa variaatioita, joissa gradienttien pituus on rajoitettu jollekin muulle välille. Joka tapauksessa koska myös paikkavektorien maksimipituus yksikkövälin sisällä on rajoitettu, pistetulolla on minimi- ja maksimiarvo.



(a) ***(b)***

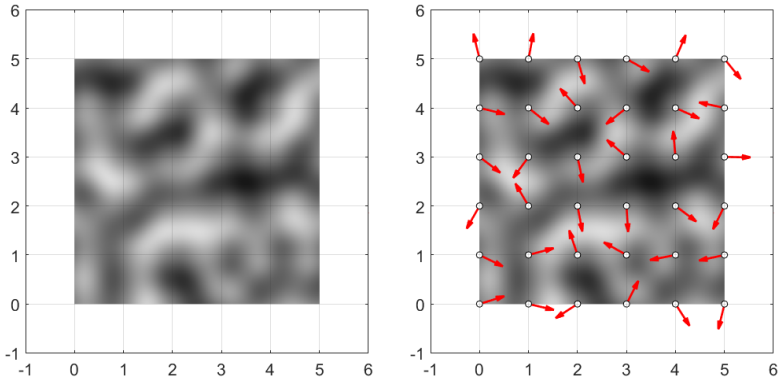
Kuva 3.5. *(a) Yksittäisen vektorin määrittämä värigradientti ja piste* $\mathbf{p}$ *jossa sitä arvioidaan. (b) Kaksiulotteisen Perlin-kohinafunktion arvoa yksikkövälin pisteessä* $P_{xy} = [x_f, y_f]$ *vastaavat paikkavektorit* $\mathbf{p}$ *(sin.) ja satunnaiset yksikkögradienttivektorit* $\mathbf{g}$ *(pun.).*

Interpolointi yksikkövälin sisällä olevien arvojen kesken toteutetaan samalla tavalla kuin

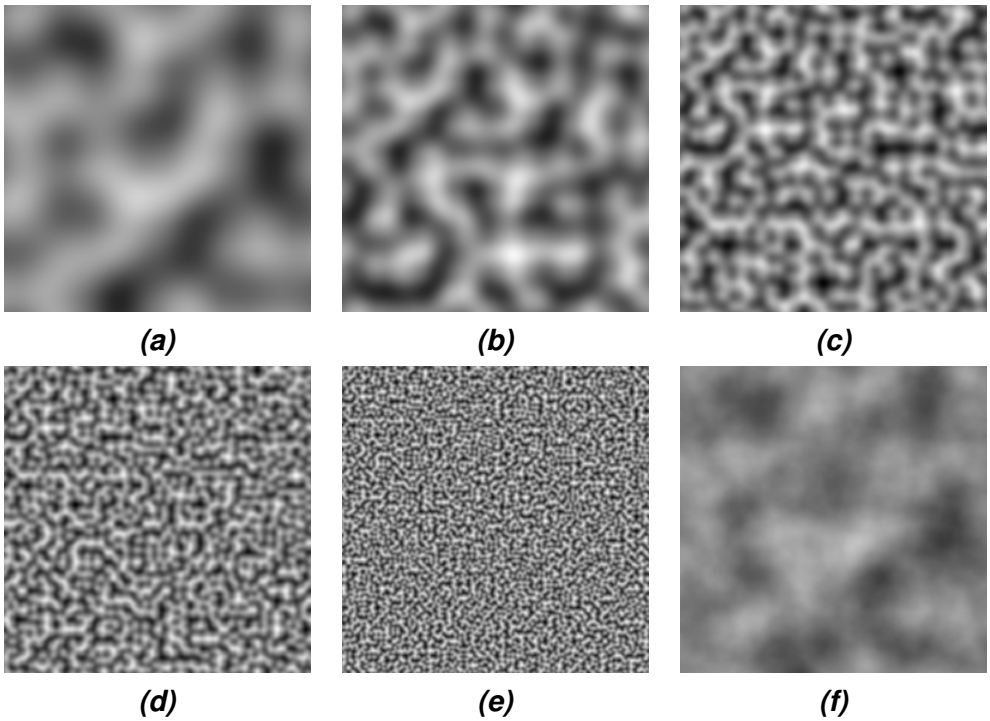
arvokohinan tapauksessa, jolloin kaksiulotteiselle Perlin-kohinalle voidaan johtaa lauseke

$$\begin{aligned} PerlinNoise(x,y) = s(1-y_f)(s(1-x_f)\cdot \mathbf{p}_{00}\cdot \mathbf{g}_{00} + s(x_f)\cdot \mathbf{p}_{10}\cdot \mathbf{g}_{10}) \\ + s(y_f)(s(1-x_f)\cdot \mathbf{p}_{01}\cdot \mathbf{g}_{01} + s(x_f)\cdot \mathbf{p}_{11}\cdot \mathbf{g}_{11}), \end{aligned} \tag{3.3}$$

missä vektorit $\mathbf{p}$ ovat kulmia vastaavat paikkavektorit ja gradienttivektorit $\mathbf{g}$ ovat toisistaan riippumattomia satunnaisia yksikkövektoreita. Kuvassa 3.6 on esitetty yhtälön 3.3 mukainen kohinaoktaavi sekä sitä vastaava gradienttivektorikenttä, josta nähdään että vektorit käytännössä määrittävät kohinafunktion kasvusuunnan matemaattisen gradientin tavoin. Esimerkki fraktaalisesta Perlin-kohinasta puolestaan on esitetty kuvassa 3.7.



Kuva 3.6. *(a) Yksi oktaavi Perlin-kohinaa ja (b) sitä vastaava gradienttivektorikenttä. (Huom. yksikkövektorien pituudet on kuvassa puolitettu selkeyden vuoksi.)*



Kuva 3.7. *(a-e) Viisi oktaavia kaksiulotteista Perlin-kohinaa ja (f) niiden fraktaalisumma ($\alpha = 1.5$, $l = 2.0$) käyttäen 5. asteen interpolaatiota.*

Kuten todettua Perlinin kohinafunktio ratkaisee arvokohinan ongelmat vain osittain. Yhtenä perustavanlaatuisena ongelmana on, että funktio saa poikkeuksetta arvon nolla kaikissa kokonaislukukoordinaateissa, mikä voidaan johtaa myös yhtälöstä 3.3 ja nähdä kuvassa 3.6. Perlin-kohinafunktion arvojakauma muistuttaa gaussista jakaumaa, mutta taajuustasossa Perlin-kohina on vain heikosti kaistarajoitettua, vaikka sen taajuudet jakautuvatkin tasaisemmin eri koordinaattisuuntiin (Lagae et al. 2010).

Muuten yleisesti kohinafunktiolle pätevät samankaltaiset lainalaisuudet kuin arvokohinalle. Esimerkiksi myös sen laskennallinen kompleksisuus on muotoa $O(2^D)$, eli algoritmi hidastuu nopeasti ulottuvuuksien määrän kasvaessa (Gustavson 2005). Lisäksi neliskanttisille hilakohina-algoritmeille tyypillisesti Perlin-kohina ei myöskään ole isotrooppista, mikä pohjimmiltaan johtuu siitä että hilassa diagonaaliset etäisyydet ovat suurempia kuin pysty- ja vaakasuuntaiset (Ebert et al. 2002, s.180).

3.2 Perlin-kohinafunktion toteutus

Ken Perlinin kohinafunktiosta on esitetty viimeisen neljän kuluneen vuosikymmenen aikana useita variaatioita. Vaikkakin edeltävässä luvussa esitellyt yleiset toimintaperiaatteet ovat edelleen voimassa, niiden käytännön toteutus ja optimointi suorituskyvyn ja muistin käytön suhteen ei ole suoraviivaista.

Perlin itse on julkaissut useita referenssitoteutuksia, joista merkittävimpiä ovat todennäköisesti sen alkuperäinen C-kielinen versio (Perlin 1985b) ja myöhempään parannuksia esittäneeseen artikkeliin (Perlin 2002b) liittyvä Java-versio (Perlin 2002a). Näihin versioihin viitataan tässä työssä klassisena ja edistyneenä Perlin-kohinana.

3.2.1 Klassinen ja edistynyt kohina

Yksi ohjelmistoteknisistä perusongelmista, jonka kaikki kohina-algoritmimt joutuvat ratkaisemaan tavalla tai toisella, on satunnaislukujen generointi paikkasidonnaisesti, eli koordinaattiyhdistelmien muuntaminen näennäisesti satunnaiseksi tulokseksi deterministisellä tavalla. Perlin-kohinan tapauksessa tämä tulos on gradienttivektori, joka sisältää D elementtiä ja jonka magnitudi on vakio.

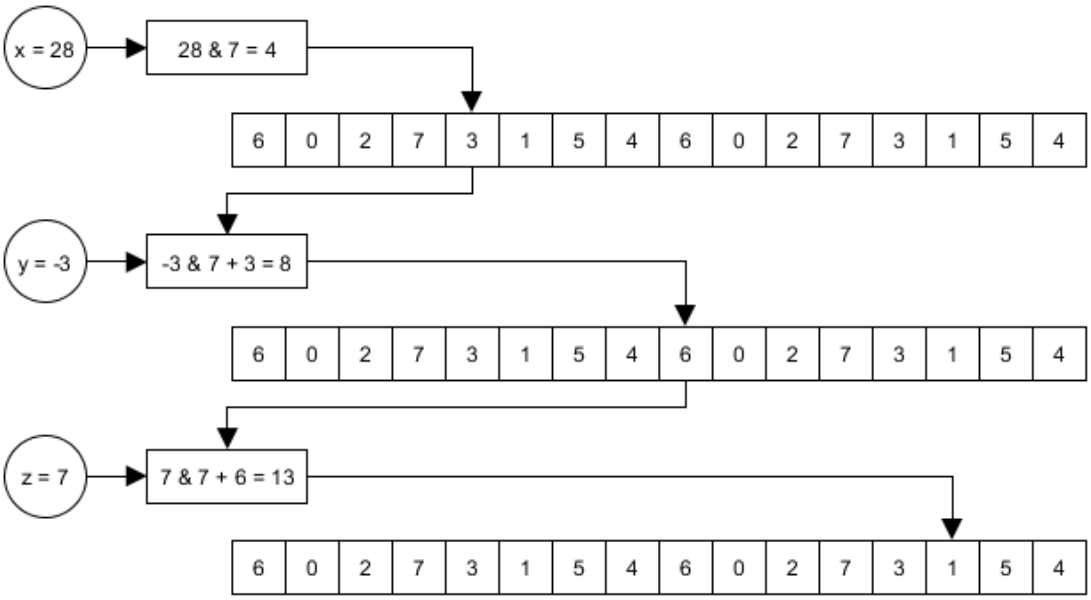
Naiivina ratkaisuna olisi käyttää koordinaatteja tavalla tai toisella satunnaislukugeneraattorin siemenlukuna ja generoida sitä kautta kokonaislukupistettä vastaava satunnaisvektori. Kaksiulotteisessa tapauksessa vektorit voitaisiin saada jakautumaan tasaisesti yksikköympyrän kehälle generoimalla tasaisesti satunnaisjakautunut kulma $\phi \in [0, 2\pi]$ ja laskemalla sitä vastaavat x- ja y-komponentit $\mathbf{g}_{xy} = [cos(\phi), sin(\phi)]$.

Käytännössä vektorikentän ei välttämättä ole pakko käyttäytyä täysin ideaalisesti ja kompromisseja voidaan tehdä laskennan nopeuttamiseksi. Perlinin (1985b) alun perin itse esittelemänä ratkaisuna oli käyttää valmiiksi generoitua satunnaista gradienttivektoritaulukkoa $\mathbf{G}$ ja sen indeksointiin hajautusfunktiota $H(x, y, z) \in [0, N_G - 1]$, missä N_G on vektoritaulukon pituus.

Alkuperäinen hajautusfunktio perustui $2N_G$-kokoiseen permutaatiotauluun $\mathbf{P}$, joka sisältää kahteen kertaan N_G-mittaisen satunnaisessa järjestyksessä olevan kokonaislukusarjan $\{0, 1, \ldots, N_G - 1\}$. Hajautusfunktio ja gradientin näennäisesti satunnainen valinta sen avulla voidaan siis esittää muodossa

$$\mathbf{g}_{xyz} = \mathbf{G}[H(x, y, z)] = \mathbf{G}[\mathbf{P}[\mathbf{P}[\mathbf{P}[mod(z, N_G)] + mod(y, N_G)] + mod(x, N_G)]], \quad (3.4)$$

missä $x, y, z \in \mathbb{Z}$. Käytännössä taulukkokooksi on kannattavaa valita jokin kahden potenssi, joka klassisen kohinan tapauksessa on $N_G = 2^8 = 256$. (Perlin ja Hoffert 1989) Tällöin jakojäännös voidaan toteuttaa nopeasti suoritettavissa olevana binäärisenä AND-operaationa $mod(x, N_G) = x \ \& \ (N_G - 1)$ (Ebert et al. 2002, s.69). Permutaatiotaulun toimintaa on havainnollistettu kuvassa 3.8.



Kuva 3.8. *Esimerkki permutaatiotaulun ($N_G = 8$) käytöstä, missä 3-ulotteisesta kokonaislukukoordinaatista $[x, y, z] = [28, -3, 7]$ lasketaan gradienttitaulun indeksi $H(x, y, z) = 1$.*

Klassisessa Perlin-kohinassa on kuitenkin useita puutteita, jotka liittyvät sekä hajautusfunktioon, että gradienttivektoreiden generointiin.

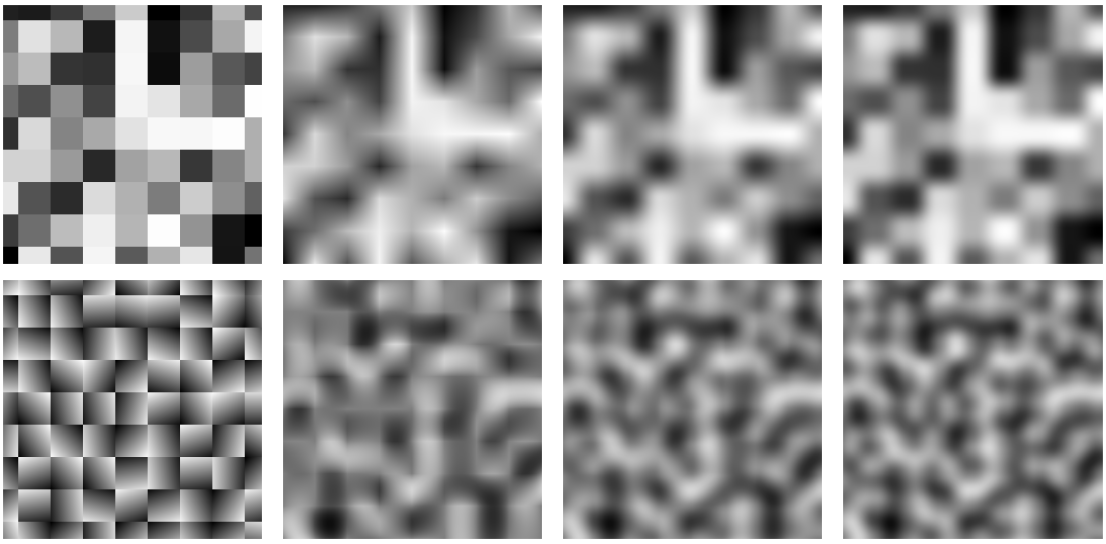
1. Perlinin referenssikoodista (1985b) ilmenee, että vektorit generoidaan valitsemalla niiden komponentit toisistaan erillisesti tasaisesta satunnaisjakaumasta ja normalisoimalla lopputulos, mistä seuraa että vektorit painottuvat diagonaalisesti yksikköpalloa vastaavan kuution kulmiin (Ebert et al. 2002, s.77).
2. Hajautusfunktion toteutus johtaa taajuustasossa selkeästi havaittavissa oleviin ei-toivottaviin aksiaalisiin riippuvuuksiin (Kensler et al. 2008).
3. Hajautusfunktiossa käytetyn jakojäännöksen ottamisen seurauksena klassinen to-

teutus toistaa itseänsä N_P-arvon määrittämän jakson välein.

4. Klassisessa Perlin-kohinassa käytettävä kolmannen asteen interpolaatiopolynomi $s_3(t) = -2t^3 + 3t^2$ johtaa epäjatkuvaan toiseen derivaattaan, kuten luvussa 3.1 todettiin.

Perlinin (2002b) esittämä edistynyt kohinafunktio ratkaisee kaksi alkuperäisen toteutuksen ongelmaa ja samalla tarjoaa pienen potentiaalisen parannuksen algoritmin suorituskykyyyn.

Kohinafunktion epäjatkuva toinen derivaatta voitiin korjata yksinkertaisesti vaihtamalla vanhan interpolaatiosplinin tilalle viidennen asteen polynomi $s_5(t) = 6t^5 - 15t^4 + 10t^3$. Ratkaisun haittapuolena interpolaatiomenetelmän monimutkaistaminen hidastaa hieman ohjelman toimintaa. (Perlin 2002b) Interpolaatiomenetelmän merkitystä lopputuloksen laatuun arvo- ja Perlin-kohinassa on havainnollistettu kuvassa 3.9.



(a) ***(b)*** ***(c)*** ***(d)***

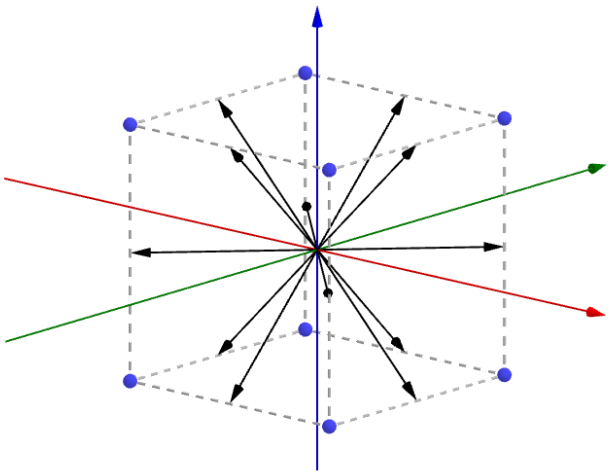
Kuva 3.9. *Yksi oktaavi arvokohinaa (ylh.) ja Perlin-kohinaa (alh.) käyttäen (a) lähimmän naapurin, (b) 1. asteen, (c) 3. asteen ja (d) 5. asteen interpolaatiota.*

Toinen ehdotettu muutos koskee gradienttivektoritaulua, joka päätettiin määritellä staattisesti, koska sen generoiminen satunnaisesti on redundanttia hajautusfunktion kanssa. Kolmiulotteisessa tapauksessa voidaan määritellä 12 kappaletta $\sqrt{2}$-pituisia vektoreita, jotka osoittavat yksikkökuution keskipisteestä särmien keskpisteisiin kuvan 3.10 mukaisesti (Perlin 2002b). Kyseessä olevat kolmiulotteiset vektorit ovat:

$$\begin{aligned}\mathbf{G}_3 = \big[\ & [0,-1,-1],[0,-1,1],[0,1,-1],[0,1,1],\\ & [-1,0,-1],[-1,0,1],[1,0,-1],[1,0,1],\\ & [-1,-1,0],[-1,1,0],[1,-1,0],[1,1,0]\ \big].\end{aligned}$$

Samaa periaatetta voidaan soveltaa myös n-ulotteisen kohinafunktion tapauksessa, jol-

loin gradienttivektoreiden pituus on yleisesti muotoa $|\mathbf{g}| = \sqrt{D-1}$. Esimerkiksi neljäulotteiselle kohinafunktiolle tämä tarkoittaisi 32 vektoria. Menetelmää voitaisiin soveltaa myös kaksiulotteiseen tapaukseen, mutta silloin gradientteja olisi vain 4 kappaletta, mikä on todennäköisesti jo liian vähän orgaanisen näköisen kohinan tuottamiseen. Sen sijaan tällöin on järkevämpää käyttää suurempaa määrää, kuten vähintään 8 tai 16 vektoria. (Gustavson 2005)



Kuva 3.10. *Edistyneen 3-ulotteisen Perlin-kohinafunktion ehdotetut 12 gradienttivektoria.*

Satunnaisgradienttien epätasainen generointi ei kuitenkaan ollut ainoana motivaationa gradienttivektoritaulun diskreettiin ja staattiseen määrittelyyn, sillä ongelma olisi pystytty ratkaisemaan muillakin tavoilla. Vielä merkittävämpänä hyötynä sen sijaan on gradienttien pistetulon laskennasta saavutettu tehokkuus, sillä valitsemalla gradienttivektoritaulun gradientit niin että niiden komponentit ovat aina -1, 0, tai $+1$, vektorien pistetulo voidaan laskea suoraan komponenttien summana poistaen yhteensä 24 kertolaskuoperaatiota (Perlin 2002b).

Kuten Perlin (2002b) itse osoittaa, tämän ominaisuuden hyödyllisyys riippuu esimerkiksi siitä onko käytettävissä laitteistokiihdytetysti toteutettua pistetulo-operaatiota.

3.2.2 Ehdotettu toteutus

Keskusteltaessa kyseessä olevien suorittimella ajettavien kohinafunktioiden suunnittelusta voidaan arvioida uudestaan tiettyjen ominaisuuksien tärkeysjärjestystä. Tässä työssä esitettävän toteutuksen motivaationa on, että siinä missä näytönohjaimessa ajettavan koodin tyypillisesti odotetaan olevan ennen kaikkea mahdollisimman nopeaa, perinteisissä ohjelmistoissa yhtä tärkeää on ohjelman monipuolisuus ja uudelleenkäytettävyys.

Tämän ohella kohinafunktion laatua haluttiin parantaa. Perusteena tälle oli, että vaikka tavoitteena onkin kohtuullisuuden rajoissa optimoida kohinageneraattorin suorituskyky, todellisuudessa suuri osa reaaliaikaisesta laskennasta, kuten renderöinnistä, tehdään joka tapauksessa laitteistokiihdytetysti.

Yleismuotoinen klassista Perlin-kohinaa mukaileva kaksiulotteinen versio kohinafunktiosta on esitetty ohjelmassa 3.1. On huomattava, että kyseessä oleva koodi ei ota kantaa hajautusfunktion toimintaan mutta olettaa, että käytettävissä on jokin gradienttivektoritaulu, jonka indeksialueen $[0, N_G - 1]$ sisältä hajautusfunktio palauttaa arvoja.

Ohjelman selkeyttämiseksi ja tiivistämiseksi luotiin hyvin yksinkertaiset struct-tyyppiset 2- ja 3-ulotteisten liukulukuvektorien luokat ja määriteltiin niille tarvittavat matemaattiset operaatiot. Kohinadatan muuta prosessointia varten samoin liukulukutyyppisille taulukoille määriteltiin joitakin matemaattisia laajennusmetodeja, vaikka kohinafunktioiden tapauksessa ohjelmisto voisi hyötyä vielä enemmän täysimittaisesta matemaattisesta matriisilaskentakirjastosta.

```
1  public static float Value2D(float x, float y, UInt64 seed) {
2
3      float x0f = (float)Math.Floor(x);
4      float xf = x - x0f;
5      Int64 x0 = (Int64)x0f;
6      Int64 x1 = x0 + 1;
7      float y0f = (float)Math.Floor(y);
8      float yf = y - y0f;
9      Int64 y0 = (Int64)y0f;
10     Int64 y1 = y0 + 1;
11
12     FloatVector2 pVector00 = new FloatVector2(xf    , yf    );
13     FloatVector2 pVector01 = new FloatVector2(xf    , yf - 1);
14     FloatVector2 pVector10 = new FloatVector2(xf - 1, yf    );
15     FloatVector2 pVector11 = new FloatVector2(xf - 1, yf - 1);
16
17     FloatVector2 gVector00 = gradientTable2D[Hash(seed, x0, y0)];
18     FloatVector2 gVector01 = gradientTable2D[Hash(seed, x0, y1)];
19     FloatVector2 gVector10 = gradientTable2D[Hash(seed, x1, y0)];
20     FloatVector2 gVector11 = gradientTable2D[Hash(seed, x1, y1)];
21
22     float gradientValue00 = Vector.Dot(gVector00, pVector00);
23     float gradientValue01 = Vector.Dot(gVector01, pVector01);
24     float gradientValue10 = Vector.Dot(gVector10, pVector10);
25     float gradientValue11 = Vector.Dot(gVector11, pVector11);
26
27     return (Interpolate.BiQuintic(xf, yf, gradientValue00,
28                                           gradientValue01,
29                                           gradientValue10,
30                                           gradientValue11)
31                                           + 1) / 2;
32 }
```

Ohjelma 3.1. *Kaksiulotteisen klassisen Perlin-kohinafunktion toteutus, jossa ulostulon arvoalue on normalisoitu välille* $[0, 1]$.

Huomautettakoon, että aiemmissa mallitoteutuksissa (Perlin 1985; 2002) kohinafunktion arvoalue on suunnilleen $[-1, 1]$, kun taas tässä tapauksessa se on skaalattu välille $[0, 1]$. Valinta eri arvoalueiden käytön välillä riippuu lähinnä siitä, kumpi on sovelluksen kannalta käytännöllisempää. Tämän ominaisuuden muuttaminen ei kuitenkaan olisi vaikeaa, mikäli sille olisi tarvetta.

Klassisen kohinan mukainen rekursiivisesti toimiva hajautusfunktio toimii koordinaattiarvojen lisäksi hyvin myös siemenluvun kanssa. Esimerkkinä tästä on ohjelman 3.2 toteutus, jossa siemenlukua käsitellään kuin koordinaattiarvoa laskemalla siitä jakojäännös ja syöttämällä se hajautusfunktioon. Toisena ilmeisenä vaihtoehtona olisi käyttää siemenlukua permutaatiotaulun sekoittamiseen, mutta ohjelman suorituskyvyn säilyttäminen tällöin vaatisi käytännössä kohinageneraattorin tilallista toteutusta.

```
1  public static Int64 Hash(UInt64 seed, Int64 x, Int64 y) {
2      return permTable[permTable[permTable[seed & 255]
3                                 + (x & 255)] + (y & 255)];
4  }
5
6  public static Int64 Hash(UInt64 seed, Int64 x, Int64 y, Int64 z) {
7      return permTable[permTable[permTable[permTable[seed & 255]
8                                 + (x & 255)] + (y & 255)] + (z & 255)];
9  }
```

Ohjelma 3.2. *Klassisen Perlin-kohinan kaksi- ja kolmiulotteiset hajautusfunktiot, joihin on lisätty tuki siemenluvulle.*

Edistyneestä kohinasta eroavasti ohjelmassa päätettiin hyödyntää perinteistä 256 vektorin mittaista gradienttitaulua, mutta klassisesta kohinasta poiketen sitä ei päätetty sekoittaa, vaan sen sisältämät yksikkövektorit on määritelty järjestyksessä staattisesti. Sen sijaan hajautusfunktion päätettiin olevan riittävä satunnaisuuden lähde.

Kuten todettua, suuri osa klassisen kohinafunktion puutteista liittyy nimenomaan sen hajautusfunktioon ja gradienttivektorien valintaan. Eräänä parannuksena voidaan mainita esimerkiksi yksi Kensler et al. (2008) esittämistä muutoksista, jossa koordinaattien välistä aksiaalista dekorrelaatiota parannetaan määrittelemällä jokaiselle koordinaattisuunnalle oma permutaatiotaulu. Menetelmä toimii erittäin hyvin lukuun ottamatta sitä, että kohinafunktio on edelleen jaksollinen 256 koordinaattiyksikön välein riippuen permutaatiotaulujen koosta.

Tästä syystä tavoitteeksi otettiin etsiä vaihtoehtoinen hajautusfunktio, jonka nopeus olisi verrattavissa alkuperäiseen, mutta joka olisi jaksoton tai sen jakso olisi 64-bittiseen kokonaislukualueeseen suhteutettuna hyvin pitkä. Tämän ohella etsitylle funktiolle erityisen tärkeitä vaatimuksia olivat jo mainittu tuki siemenluvulle ja epäsymmetrisyys, eli funktiolle on oltava voimassa $H(x, y) \neq H(y, x)$. Ohjelmassa 3.3 on esitetty kaksi kohtalaisen hyvin toimivaa hajautusfunktiota kaksi- ja kolmiulotteiselle kohinalle.

```
1  public static UInt64 Hash(UInt64 seed, Int64 x, Int64 y) {
2      unchecked {
3          x = (x * 22649707571907919) ^ (x >> 32);
4          y = (y * 32538617539989569) ^ (y >> 32);
5          return ((((UInt64)(x ^ y) ^ seed)
6                  * 6352870822070075357)) >> 56;
7      }
8  }
9
10 public static UInt64 Hash(UInt64 seed, Int64 x, Int64 y, Int64 z) {
11     unchecked {
12         x = (x * 22649707571907919) ^ (x >> 32);
13         y = (y * 32538617539989569) ^ (y >> 32);
14         z = (z * 41263188386085229) ^ (z >> 32);
15         return ((((UInt64)(x ^ y ^ z) ^ seed)
16                 * 6352870822070075357)) >> 56;
17     }
18 }
```

Ohjelma 3.3. *Vaihtoehtoiset hajautusfunktiot kaksi- ja kolmiulotteiselle Perlin-kohinalle.*

Yleisesti yksinkertaisissa ei-kryptografisissa hajautusfunktioissa voidaan päästä kohtuullisiin lopputuloksiin jo muutamien aritmeettisten ja binääristen operaatioiden kautta. Tässä tapauksessa koordinaatit sekoitettiin siirtämällä niitä bittitasossa ja ottamalla XOR-operaatiot niiden ja alkuperäisen arvon kanssa. Lisäksi tulokset kerrotaan useassa kohdassa suurilla alkuluvuilla, jolloin käytetään hyväksi etumerkillisten (kahden komplementin) ja etumerkittömien kokonaislukujen ylivuotoa, minkä takia operaatiot suoritetaan C#:n unchecked-lohkon sisällä.

Vaikka XOR-operaatiot ovatkin symmetrisiä, niiden avulla voidaan edelleen yhdistää tulokset erittäin tehokkaasti, kunhan niitä edeltää joukko epäsymmetrisiä operaatioita. Lopuksi tämän tulosten yhdistämisen jälkeen saatava arvo kerrotaan vielä kertaalleen suurella alkuluvulla ja siitä palautetaan kahdeksan eniten merkitsevää bittiä.

Siemenlukuna toimii tässä tapauksessa hyvin muun muassa järjestelmän kellonaika, jota yleisesti käytetään muutenkin satunnaislukugeneraattoreissa. Tällaisen aikaleiman vähiten merkitsevät bitit muuttuvat nopeasti, mutta koska yhdistetty tulos kerrotaan niin suurella arvolla, muutokset vähiten merkitsevissä biteissä vaikuttavat suuresti myös eniten merkitseviin.

Sivuhuomautuksena liukulukujen laskennassa päätettiin käyttää 32-bittistä tarkkuutta käytännön syistä, jotka eivät ole tutkimuksen aiheen kannalta olennaisia lukuunottamatta kasvavaa muistinkäyttöä. Mikäli täydelle 64-bittiselle tarkkuudelle olisi sovelluksessa tarvetta, vaadittavan refaktoroinnin tekeminen olisi silti triviaalia.

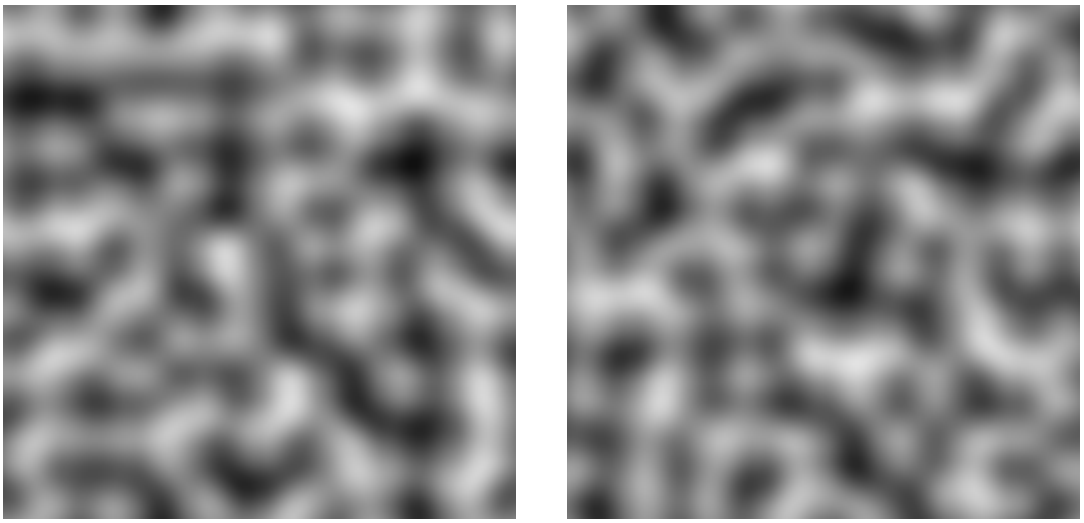
4 TULOKSET

Työssä esiteltiin vaihtoehtoiset versiot kaksi- ja kolmiulotteisesta Perlin-kohinafunktiosta, jotka perustuvat uuteen hajautusfunktioon ja klassisen kohinan mukaiseen valmiiksi laskettuun gradienttitauluun. Tässä luvussa kaksiosaisena tavoitteena on sekä tarkastella toteutettujen kohinafunktioden teknisiä ja laadullisia ominaisuuksia, että demonstroida kohinan sovellustapoja muutamien esimerkkien kautta.

4.1 Kohinafunktion suorituskyky ja laatu

Perlin-kohinafunktion aiemmat referenssitoteutukset (Perlin 1985; 2002) ovat keskittyneet vahvasti optimoimaan funktion suorituskykyä äärimmilleen tehden joitakin kompromisseja kohinan laadun suhteen. Kontrastina tähän tässä työssä esitetyn uuden vaihtoehtoisen toteutuksen keskeisinä kriitereinä olivat sekä suorituskyky, että laatu, joista molempia näkökulmia on syytä tarkastella tarkemmin.

Pinnallista visuaalista tarkastelua voidaan tehdä suoraan kohinafunktioiden tuottamille kohinakuville, joista esimerkit on esitetty kuvassa 4.1.



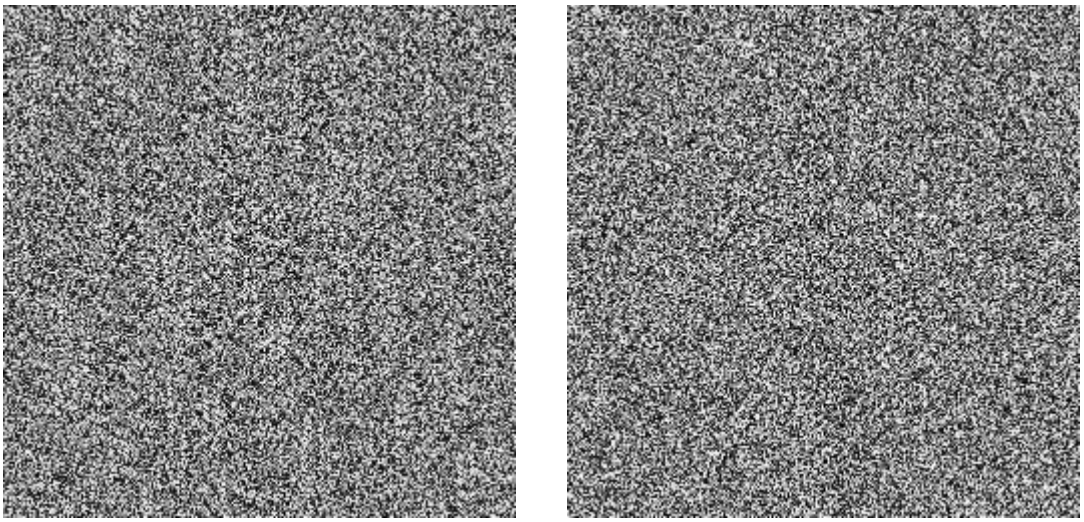
(a) ***(b)***

Kuva 4.1. *Yksi oktaavi (a) klassista ja (b) uutta kaksiulotteista Perlin-kohinaa koordinaattivälillä* $x, y \in [-128, 127]$*.*

Näennäisesti molemmat kohinafunktiot tuottavat käyttökelpoista kohinadataa, eikä käytännön sovelluksissa todennäköisesti olisi helppoa erottaa niitä toisistaan. Toisin kuin klassisessa kohinassa, uudessa Perlin-kohinafunktiossa gradienttitaulu toteutettiin niin, että vektorit ovat jakautuneet tasaisesti yksikköympyrän kehälle, minkä pitäisi tuottaa hie-

man parempia tuloksia.

Kuten todettua, kohinageneraattorin ytimessä on sen käyttämä hajautusfunktio, joka klassisessa kohinassa oli toteutettu permutaatiotaulun avulla. Sen korvanneella uudella hajautusfunktiolla on joitakin etuja, kuten jaksottomuus ja jopa hieman pienempi muistijalanjälki. Olennaisinta ovat kuitenkin silti hajautusfunktion satunnaisuusominaisuudet, joita tutkittiin laskemalla hajautusfunktioiden arvoja kokonaislukukoordinaatistossa ja tuottamalla tuloksista harmaasävykuva, jonka ideaalisessa tapauksessa pitäisi olla valkoista kohinaa kaikkialla kokonaislukuavaruudessa. Tulokset on esitetty kuvassa 4.2.



(a) ***(b)***

Kuva 4.2. *Hajautusfunktioiden ulostuloarvot $H(x, y) \in [0, 255]$ kaksiulotteisessa koordinaatistossa välillä $x, y \in [-128, 127]$ (a) klassisen ja (b) uuden funktion tapauksessa.*

Tuloskuvista nähdään, että uusi hajautusfunktio tuottaa jopa hieman parempaa satunnaisuutta kuin vanha, joka vaikuttaa tuottavat heikosti havaittavia kuvioita. On kuitenkin syytä huomauttaa, että klassisen hajautusfunktion laatu riippuu sen käyttämästä permutaatiotaulusta, eli tuloskuva voisi näyttää erilaiselta toista taulua käyttäen. Näiden tulosten pohjalta on vaikeaa arvioida, olisiko mahdollista löytää sellainen permutaatiotaulu, joka tuottaisi enemmän satunnaisilta vaikuttavia tuloksia. Klassisen hajautusfunktion jaksollisuus permutaatiotaulun koon rajoittaman lukualueen välein on kuitenkin edelleen ylitsepääsemätön ongelma.

Näin ollen laadullisesti uutta kohinafunktiota voidaan pitää parempana, mutta sen lisäksi tärkeää on vertailla uuden funktion suorituskykyä alkuperäiseen. Tämän johdosta klassista ja uutta kohinaa testattiin 50 000 iteraation yksisäikeisessä suorituskykytestissä, jonka tulokset on esitetty taulukossa 4.1

Taulukko 4.1. *Klassisen ja uuden kohinan suorituskykytestin tulokset 50 000 iteraatiosta yhden 256x256-oktaavin Perlin-kohinaa (Suoritin: Intel Core i5 4690K @ 4.2 GHz)*

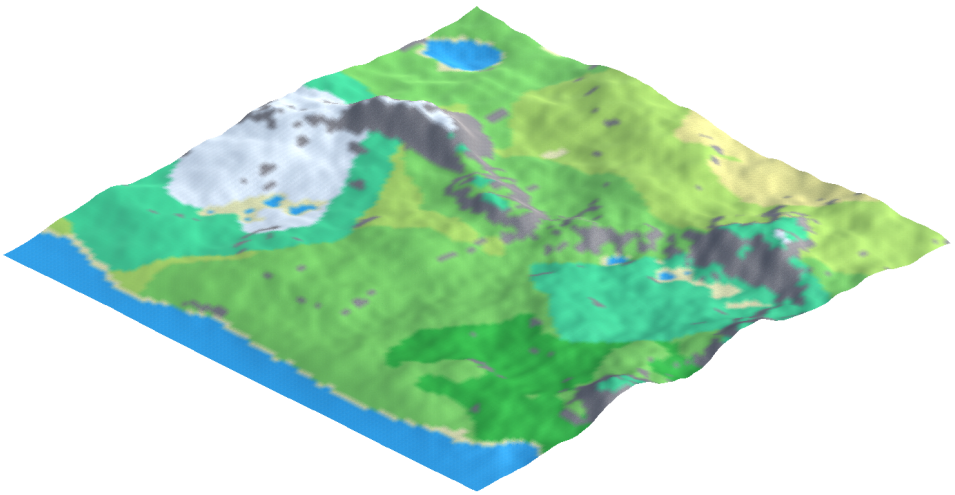
	Aika yht. (s)	Aika/iteraatio (ms)	FPS
Klassinen kohina	752	15.1	66.4
Uusi kohina	779	15.6	64.1

Testien perusteella uusi kohinafunktio on melkein yhtä suorituskykyinen kuin sen klassinen vastine laskenta-ajan ollessa noin 97 prosenttia alkuperäisestä käytetyssä testiympäristössä. Vaikka reaaliaikaisen kohinan generointi onkin usein järkevämpää toteuttaa laitteistokiihdytetysti, voidaan suorittimella ajettuja testejä pitää suuntaa antavina.

4.2 Käytännön esimerkkisovelluksia

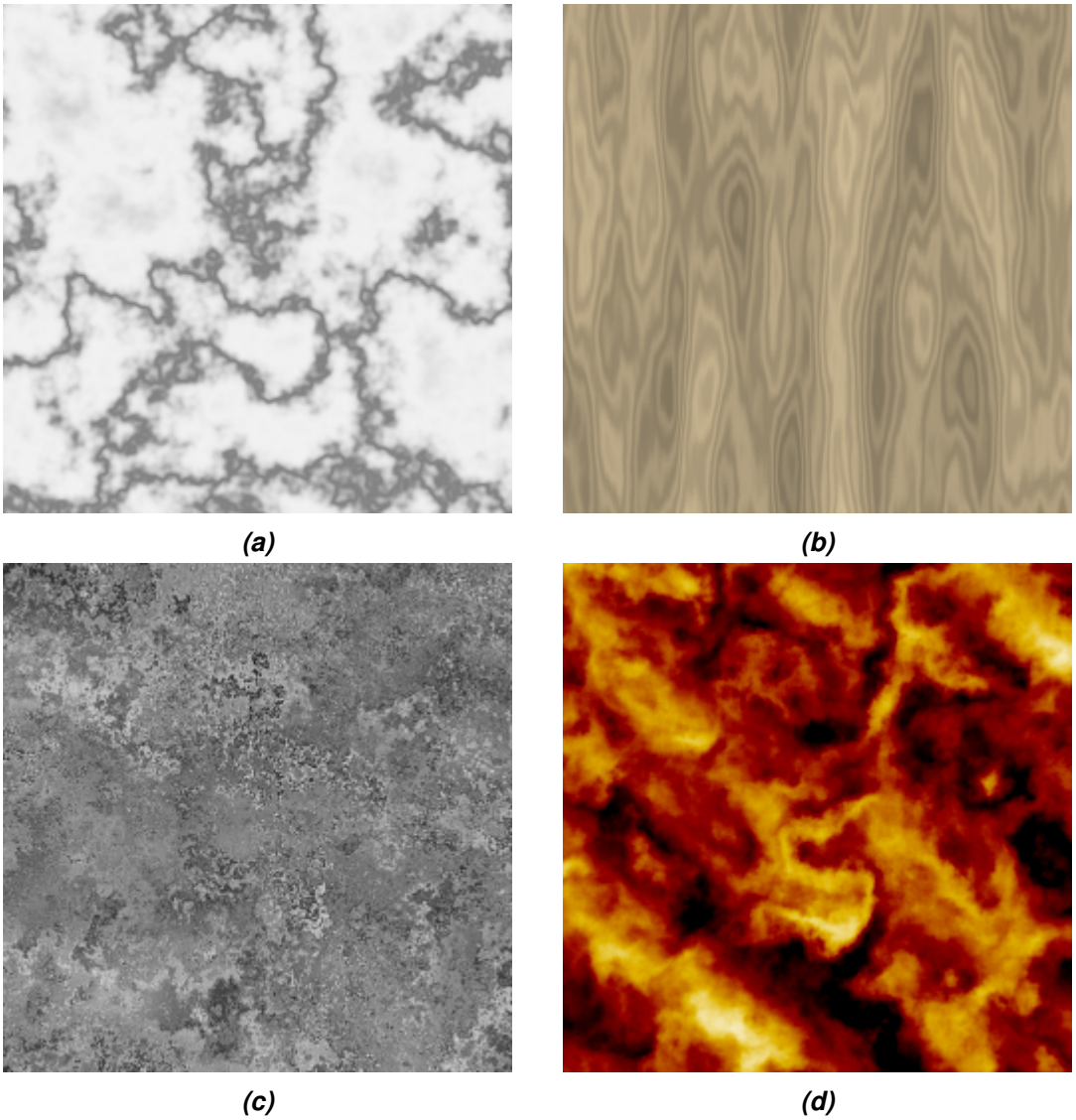
Käytännön sovelluksissa perusongelmana on tyypillisesti tuottaa sellainen kohinageneraattori, joka tuottaa spesifejä luonnonilmiöitä jäljittelevvää kohinadataa. Kyseessä voi olla esimerkiksi tekstuuri, maaston korkeuskartta, volumetrinen kaasupilvi, 3D-mallin animaatio, hiukkasen liikerata tai mitä tahansa muuta. Syvällinen keskustelu näistä sovellusalueista ei ole tämän työn piirissä, mutta sen sijaan tavoitteena on esittää joitakin esimerkkejä, joissa yhdistellään monia manipulointimenetelmiä toivotun lopputuloksen aikaan saamiseksi.

Yksi klassinen kohinafunktioiden sovelluskohde on maaston tai muun proseduraalisen geometrian generointi. Tässä tapauksessa kiinnostavia tuloksia voidaan saavuttaa jo yksinkertaisen fraktaalisen kohinageneraattorin avulla. Kuvassa 4.3 on esitetty melko suoraviivaisesti kaksiulotteisesta kohinadatasta generoitu kartta, jossa kohinaa käytetään paitsi maaston korkeuden, myös osittain lämpötilan ja ilmankosteuden määrittämiseen.



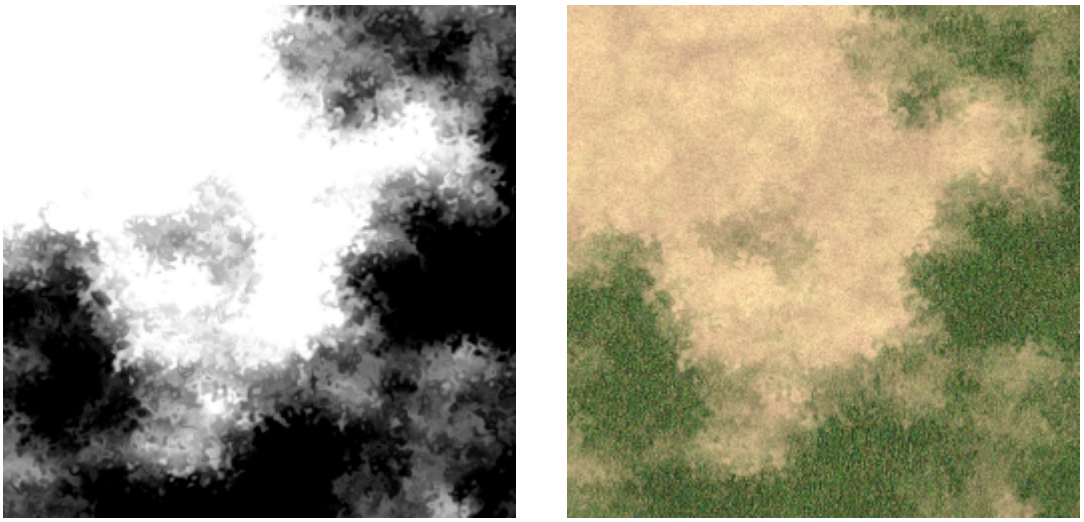
Kuva 4.3. *Satunnaisesti generoidusta maaston korkeutta, lämpötilaa ja kosteutta mallintavasta kohinadatasta luotu kartta.*

Kuva 4.3 on esimerkki tyypillisestä sovelluksesta, joka yhdistää proseduraalista geometriaa ja teksturointia. Kuvassa 4.4 puolestaan on esitetty erilaisten pisteoperaatioiden ja siirtovääristymäefektien kautta generoituja monimutkaisempia proseduraalisia luonnon tekstuureja ja efektejä. Voidaan huomata, että melkein mitä tahansa satunnaisia kuvioita pystytään jäljittelemään. Esimerkit ovat kaksiulotteisia, mutta niiden muuntaminen kolmiulotteisiksi tai animoiminen olisi myös melko vaivatonta.



Kuva 4.4. *(a) Marmoria, (b) puuta, (c) kiveä ja (d) tulta jäljitteleviä kohinatekstuureja ja -efektejä hyödyntäen pisteoperaatioita ja siirtovääristymiä.*

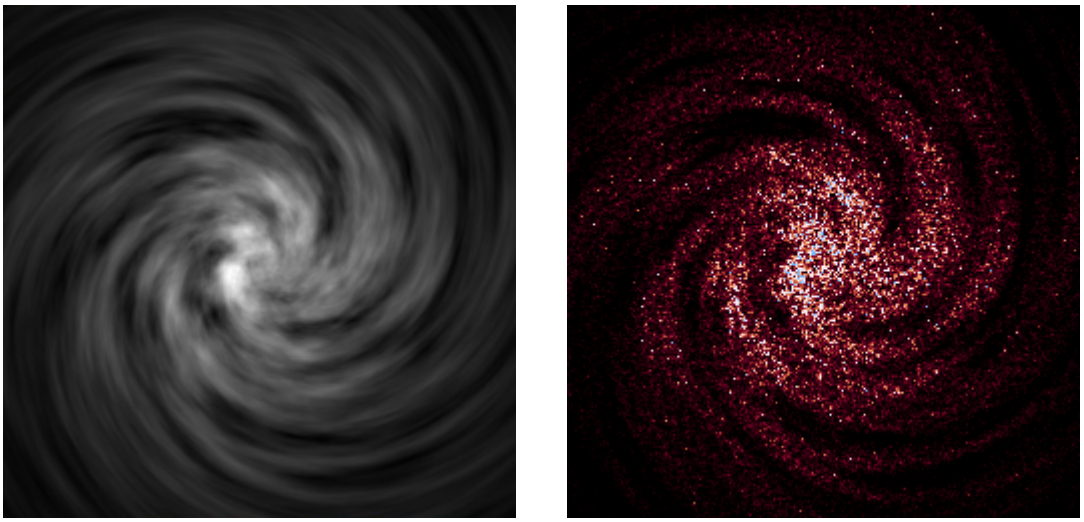
Kohinafunktioita voidaan käyttää myös eri tekstuurien tai muiden komponenttien yhdistämiseen pisteittäisten maskioperaatioiden avulla. Kuva 4.5 sisältää esimerkin, jossa kaksi proseduraalista kohinatekstuuria sulautetaan toisiinsa käyttäen kolmatta maskina toimivaa kohinafunktiota. Huomautettakoon, että komponenttitekstuurien ei ole välttämätöntä olla synteettisesti generoituja, vaan menetelmää voidaan käyttää mihin tahansa kuvadataan. Maskioperaatiot voivat olla hyödyllisiä myös maaston generoinnissa, koska niitä voidaan käyttää eri maastotyyppien toisiinsa sulauttamiseen.



(a) ***(b)***

Kuva 4.5. *(a) Maski, joka on generoitu vääristämällä kohinafunktiota ja lisäämällä sen kontrastia, ja (b) tuloskuva jossa proseduraaliset hiekka- ja ruohotekstuurit on sulautettu toisiinsa.*

Kohinan avulla voidaan siis generoida dataa melkein mihin tahansa tarkoitukseen, jos jäljiteltävä ilmiö vain on jollain tavalla simuloitavissa kohinan avulla. Yksi vähemmän suoraviivainen esimerkiksi on kuvassa 4.6 kohinan avulla mallinnettu galaksi, jossa kohinaa on käytetty tähtien esiintymätodennäköisyyden ja massan odotusarvon funktiona.

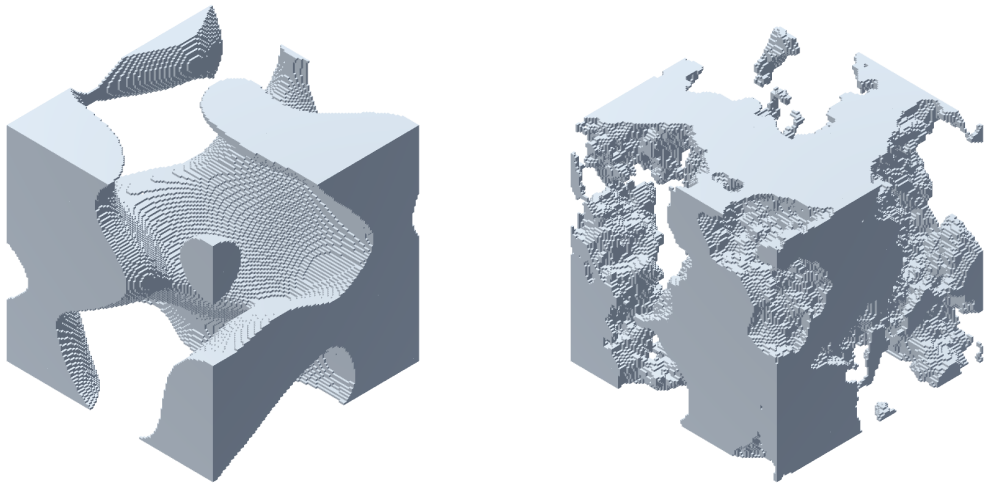


(a) ***(b)***

Kuva 4.6. *(a) Polaarikoordinaatistossa säteen funktiona kierretty (siirtovääristynyt) ja vaimennettu kohinafunktio ja (b) sen pohjalta generoitu yksittäisiä tähtiä sisältävä galaksi.*

Viimeisenä tässä luvussa käsitellään esimerkkejä kolmiulotteisten kohinageneraattorien hyödyntämisestä. Kuvassa 4.7 on havainnollistettu kolmiulotteista Perlin-kohinaa vokselipohjaisen renderöijän avulla. On syytä huomauttaa, että kolmiulotteisen kohinan visualisoimiseksi renderöijälle on jouduttu asettamaan kiinteän ja läpinäkyvän alueen raja-

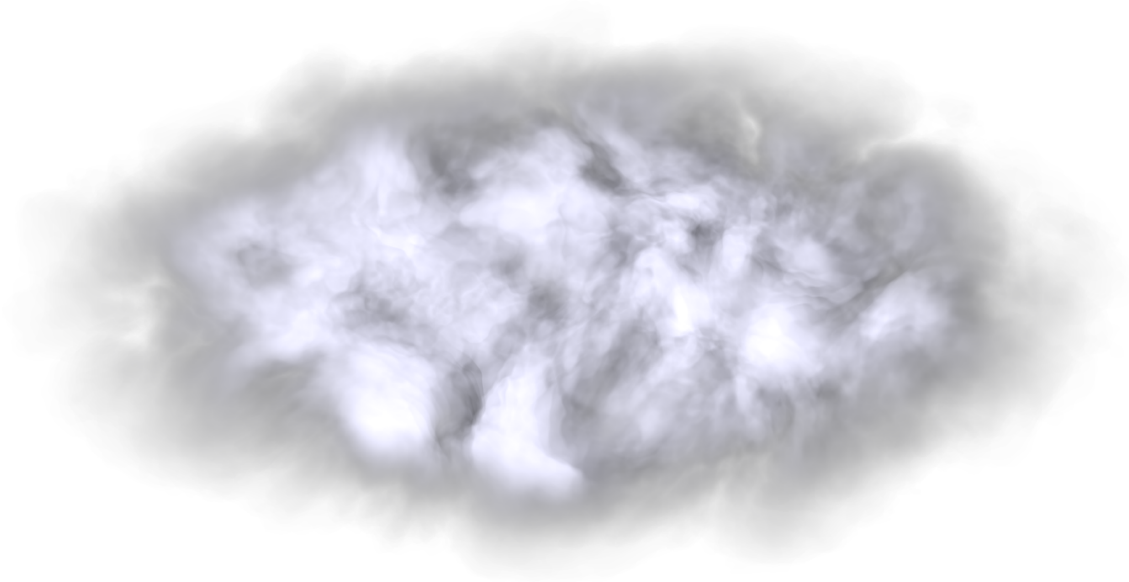
arvoksi mielivaltainen lukuarvo, joka tässä tapauksessa on käytetyn kohinafunktion arvoalueen keskikohta ($v = 0.5$).



(a) ***(b)***

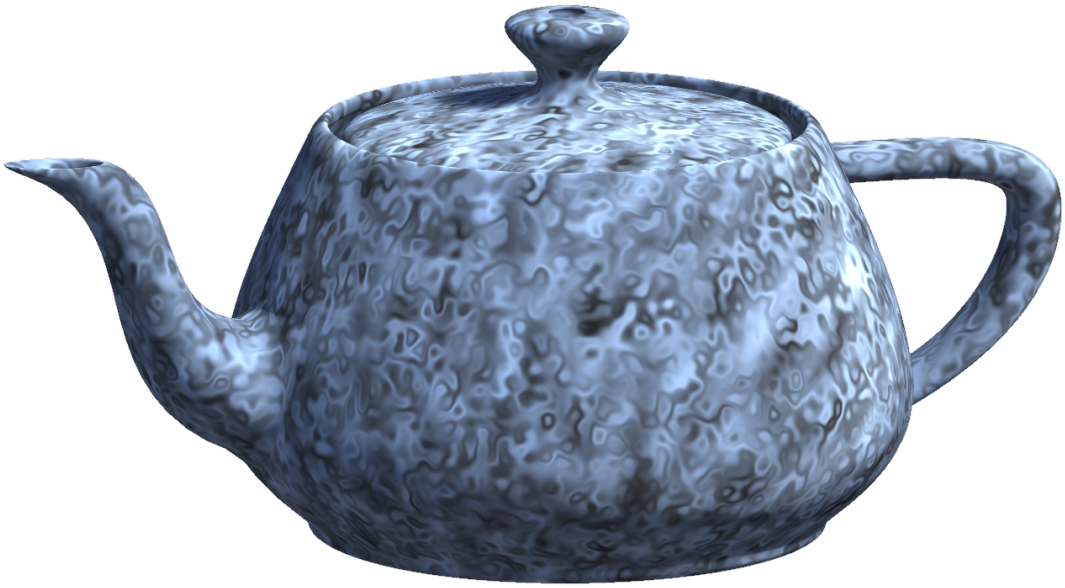
Kuva 4.7. *Vokseleina renderöity läpileikkaus kolmiulotteisen uuden Perlin-kohinafunktion (a) yhdestä oktaavista ja (b) viidestä oktaavista fraktaalista kohinaa ($l = 2.0$, $a = 1.0$).*

On olemassa myös toinen, luontaisempi esitystapa kolmiulotteisille kohinafunktioille. Volumetristen renderöintimenetelmien avulla kohinaa voidaan hyödyntää läpikuultavien kaasujen tai muiden huokoisten materiaalien tuottamisessa kuvan 4.8 esimerkin mukaisesti. Tässä tapauksessa renderöinti toteutettiin säteenjäljitykseen (engl. *ray tracing*) pohjautuvan HLSL-varjostimen (engl. *shader*) avulla.



Kuva 4.8. *Säteenjäljitykseen perustuvan HLSL-ohjelman avulla renderöity volumetrinen pilvi. Kyseessä olevassa fraktaalisessa kohinageneraattorissa on hyödynnetty turbulenssia ja siirtoväristymiä.*

Lopuksi kuvassa 4.9 nähdään toisena HLSL-ohjelmana toteutettu kolmiulotteinen tekstuuri, jota on sovellettu teekannun muotoiseen 3D-malliin. Menetelmä perustuu siihen, että kolmiulotteisen kohinageneraattorin arvoja näytteistetään kappaleen pinnan pisteissä, jolloin saatavasta kohina-arvosta muodostetaan pintapisteen RGB-väriarvo. Tällöin tuloksena saatava tekstuuri on matemaattisessa mielessä täydellinen kuvaus siitä, että kappale olisi "veistetty"yhdestä kiinteästä volyymista sen sijaan että tekstuuria olisi tarpeen projisoida kappaleen pinnalle keinotekoisesti UV-kartan avulla.



Kuva 4.9. *HLSL-ohjelmana toteutetun siirtovääristyneen Perlin-kohinafunktion avulla teksturoitu 3D-objekti.*

Kohinageneraattorit ovat hyvin käytännöllinen tapa tuottaa kolmiulotteisia tekstuureja, koska korkearesoluutioisen 3D-tekstuurin kokonaisena säilyttäminen veisi suuren määrän tallennustilaa ja muistia. Muina etuina on, että kohinafunktiota käytettäessä kappaleen muodolla tai kameran resoluutiolla ei myöskään ole väliä. Lisäksi kohinafunktioiden parametrisen luonteen takia erilaisten dynaamisten efektien toteuttaminen on tällöin helpompaa. Menetelmään viitataan yleisesti *kiinteänä kohinana* (engl. *solid noise*) (Ebert et al. 2002, s.10).

5 YHTEENVETO

Proseduraalisia kohinageneraattoreita on tämän työn aikana tarkasteltu kolmesta eri näkökulmasta, joita olivat niiden teoria, toteutus ja soveltaminen. Näiden kohtien kautta on luotu kattava yleistason kuvaus aihepiiristä.

Kohinafunktiot määriteltiin pohjimmiltaan satunnaislukugeneraattoreina, joilla on tiettyjä erityispiirteitä, kuten paikkasidonnaisuus, ja mahdollisesti korrelaatio-ominaisuuksia jos kyse on koherentista kohinasta. Tällaisen kohinafunktion tuottama data sisältää silloin moniulotteisia rakenteita, mikä erottaa sen tavallisesta satunnaisuudesta.

Tutkimalla kohinafunktioita taajuustasossa voitiin todeta, että tietyn kokoisia rakenteita tuottava kohinafunktio on taajuussisällöltään kaistarajoitettu. Tästä johdettiin fraktaalisen kohinageneraattorin konsepti, missä skaalaamalla perusmuotoisia kohinafunktioita ne pystyttiin yhdistämään yhdeksi summafunktioksi, jonka kokonaistaajuusjakauma oli helposti hallittavissa. Tätä nimitettiin fraktaalisummaksi, koska se sisälsi erikokoisia yksityiskohtia, joissa sama perusfunktio toistui itsessänsä.

Fraktaalinen kohina on kuitenkin vain yksi kohinan generoinnin perusmenetelmä, joka voidaan yhdistää muihin, edistyneisiin kohinan manipulointimenetelmiin. Näiksi tunnistettiin ainakin pisteoperaatiot, turbulentti kohina, siirtovääristymät ja animoitu kohina. Nämä menetelmät ovat myös yhdistettävissä eri tavoin, minkä avulla pystytään luomaan vielä vaikuttavampia kohinaefektejä, joita olisi hyvin vaikea jäljitellä pelkkien perusmuotoisten funktioiden avulla.

Teoreettisten käsitteiden ja konseptien lisäksi työssä tutkittiin joitakin käytännön kohinaalgoritmeja, joiden kautta voidaan aproksimoida ideaalisia kaistarajoitettuja kohinafunktioita. Kaikista yksinkertaisimmaksi algoritmiksi tunnistettiin arvokohina, jonka voitiin kuitenkin todeta tuottavan laadullisesti melko heikkoja tuloksia joitakin käyttötapoja lukuunottamatta.

Arvokohinan jälkeen siirryttiin käsittelemään Perlin-kohinaa, jossa hyödynnetään satunnaisia gradienttivektorikenttiä kohinan tuottamiseen. Menetelmän avulla oli mahdollista päästä paljon orgaanisempiin lopputuloksiin, vaikka algoritmin toimintaperiaatteista johtuen sillä onkin edelleen joitain perustavanlaatuisia puutteita ideaalisiin kaistarajoitettuihin kohinafunktioihin verrattuina.

Lisäksi Perlin-kohinan aiemmat toteutukset kärsivät joistakin teknisistä rajoitteista, joiden ratkaisemiseen työssä haluttiin kiinnittää huomiota. Keskusteluun sisälletyt vertailut rajattiin niin kutsuttuihin klassiseen ja edistyneeseen kohinaan, joista jälkimmäinen ratkaisi ongelmat osittain mutta oli hajautusfunktionsa puolesta edelleen paranneltavissa muun muassa sen jaksollisuuden takia.

Työssä esitettiin Perlin-kohinafunktion vaihtoehtoinen toteutus, jossa hajautusfunktio korvattiin uudella aritmeettisten ja binääristen operaatioiden avulla toteutettuna ratkaisuna. Lopputuloksena saadun Perlin-kohinafunktion voitiin nähdä olevan laadullisesti jossain määrin parempaa ja suorituskyvyltään 97 prosenttia vastaavan klassisen kohinan laskentanopeudesta.

Lopuksi työssä demonstroitiin joitakin kohinan käyttökohteita proseduraalisen tietokonegrafiikan ja ympäristöjen generoinnin näkökulmasta. Näin voitiin luoda ennalta generoituja tai reaaliaikaisia tekstuureja ja efektejä, sekä yksinkertaisia kohinan avulla simuloituja potentiaalisesti rajoittamattoman laajoja ympäristöjä. Kohinaan pohjautuvalla proseduraalisella tietokonegrafiikalla voitiin osoittaa olevan myös suuri määrä potentiaalisia hyötyjä perinteisiin renderöintimenetelmiin nähden.

Näennäisestä laajuudestansa huolimatta työssä kyettiin saavuttamaan vain pinnallinen kosketus kohinageneraattoreiden toimintaan ja mahdollisiin sovellustapoihin. Tulevaisuudessa laitteistojen kasvava suorituskyky ja muu teknologinen kehitys voivat edelleen laajentaa kohinafunktioiden merkitystä tietokonegrafiikassa, sisällön generoinnissa ja simulaatioissa. Onkin hyvin mahdollista, että kohinaa ja muita proseduraalisia menetelmiä tullaan näkemään yhä enenevissä määrin näillä sovellusaloilla.

LÄHDELUETTELO

Balboa, R. ja N. Grzywacz (joulukuu 2003). Power spectra and distribution of contrasts of natural images from different habitats. *Vision research* 43, 2527–37. DOI: 10.1016/S0042-6989(03)00471-1.

Ball, P. (1999). *The Self-Made Tapestry - Pattern formation in nature*. 1. painos. Oxford University Press. ISBN: 978-0198502449.

Blender, R., X. Zhu ja K. Fraedrich (2011). Observations and modelling of 1/f-noise in weather and climate. *Advances in Science and Research* 6.1, 137–140. DOI: 10.5194/asr-6-137-2011.

Cook, R. L. ja T. DeRose (heinäkuu 2005). Wavelet Noise. *ACM Trans. Graph.* 24.3, 803–811. ISSN: 0730-0301. DOI: 10.1145/1073204.1073264.

Ebert, D. S., F. K. Musgrave, D. Peachey, K. Perlin ja S. Worley (2002). *Texturing and Modeling: A Procedural Approach*. 3. painos. Morgan Kaufmann Publishers Inc. ISBN: 1558608486.

Gagnon, J.-S., S. Lovejoy ja D. Schertzer (lokakuu 2006). Multifractal earth topography. *Nonlinear Processes in Geophysics* 13.5, 541–570.

Gustavson, S. (2005). *Simplex Noise Demystified*. Tekninen raportti. Linköping University. DOI: 10.13140/RG.2.1.3369.6488.

Havlin, S., S. V. Buldyrev, A. L. Goldberger, R. N. Mantegna, S. M. Ossadnik, C.-K. Peng, M. Simons ja H. E. Stanley (1995). Fractals in biology and medicine. *Chaos, Solitons & Fractals* 6. Complex Systems in Computational Physics, 171 –201. ISSN: 0960-0779. DOI: 10.1016/0960-0779(95)80025-C.

Kensler, A., A. Knoll ja P. Shirley (2008). *Better Gradient Noise*. Tekninen raportti. SCI Institute.

Lagae, A., S. Lefebvre, R. Cook, T. Derose, G. Drettakis, D. Ebert, J. P. Lewis, K. Perlin ja M. Zwicker (2010). A Survey of Procedural Noise Functions. *Computer Graphics Forum* 29. DOI: 10.1111/j.1467-8659.2010.01827.x.

Mandelbrot, B. B. (1983). *The fractal geometry of nature*. 3. painos. W. H. Freeman ja Comp. ISBN: 0-7167-1186-9.

Olano, M. (2005). Modified Noise for Evaluation on Graphics Hardware. Teoksessa: *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*. HWWS '05. ACM, 105–110. ISBN: 1-59593-086-8. DOI: 10.1145/1071866.1071883.

Olano, M., J. Hart, W. Heidrich ja M. McCool (2002). *Real-Time Shading*. A. K. Peters, Ltd. ISBN: 1568811802.

Perlin, K. ja E. M. Hoffert (heinäkuu 1989). Hypertexture. *SIGGRAPH Comput. Graph.* 23.3, 253–262. ISSN: 0097-8930. DOI: 10.1145/74334.74359.

Perlin, K. (1985a). An Image Synthesizer. *SIGGRAPH Comput. Graph.* 19.3, 287–296. ISSN: 0097-8930. DOI: 10.1145/325165.325247.

Perlin, K. (1985b). *Noise and Turbulence*. URL: https://mrl.nyu.edu/~perlin/doc/oscar.html#noise (viitattu 16. 02. 2019).

– (2002a). *Improved Noise Reference Implementation*. URL: https://mrl.nyu.edu/~perlin/noise/ (viitattu 16. 02. 2019).

– (2002b). Improving Noise. *ACM Trans. Graph.* 21.3, 681–682. ISSN: 0730-0301. DOI: 10.1145/566654.566636.

– (2004). Implementing Improved Perlin Noise. Teoksessa: *GPU Gems*. Toim. R. Fernando. 1. painos. Addison-Wesley. Luku 5.

Ramstedt, R. ja J. Smed (syyskuu 2016). Midpoint Displacement in Multifractal Terrain Generation. Teoksessa: *Proceedings of the 17th International Conference on Intelligent Games and Simulation (Game-On 2016)*.

Tolhurst, D. J., Y. Tadmor ja T. Chao (1992). Amplitude spectra of natural images. *Ophthalmic & physiological optics : the journal of the British College of Ophthalmic Opticians (Optometrists)* 12.2, 229.

West, B. J. ja M. Shlesinger (1990). The Noise in Natural Phenomena. *American Scientist* 78.1, 40–45. ISSN: 00030996.