Kolmiuloitteisten ympäristöjen pintamuotojen proseduraalinen generointi videopeleissä

Turun yliopisto
Tietotekniikan laitos
LuK-tutkielma
Tietojenkäsittelytiede
Huhtikuu 2025
Evert Mäkinen

TURUN YLIOPISTO
Tietotekniikan laitos

Evert Mäkinen: Kolmiuloitteisten ympäristöjen pintamuotojen proseduraalinen generointi videopeleissä

LuK-tutkielma, 24 s.
Tietojenkäsittelytiede
Huhtikuu 2025

Peliympäristö on videopeleissä keskeinen elementti. Ympäristöjen koko ja yksityiskohtainen realismi on noussut pelien kehittyessä, mutta näiden maailmojen suunniteleminen manuaalisesti vie paljon rahaa ja aikaa. Ympäristöjen luonti algoritmisesti proseduraalisella generoinnilla vähentää manuaalisen suunnitelun määrää pelin kehityksestä ja samalla mahdollistaa loputtoman skaalan ympäristöille.
Tutkielman tarkoituksena on tarkastella kolmiuloitteisten peliympäristöjen proseduraalista generointia tieteellisten tutkimusten avulla kirjallisuuskatsauksena. Tutkielmassa käsitellään kolmiuloitteisten polygoniverkkojen luontia, sekä pintamuotoja vastaavien korkeuskarttojen generointia proseduraalisesti. Tutkielmassa käydään läpi proseduraalisen generaation standarditekniikoista kehittyneempiin koneoppimista hyödyntäviin metodeihin ja tarkastallaan niiden hyötyjä sekä haittoja.
Tutkielmassa todetaan, että proseduraalinen generaatio mahdollistaa paljon videopeleissä, mutta siitä on vielä melko vähäisesti tieteellistä tutkimusta. Koneoppimisen hyödyntäminen tuottaa realistisia tuloksia, mutta generoituja tuloksia ei tämänhetkisillä toteutuksilla ole pystytty käyttämään peleissä generoitujen korkeuskarttojen saumattomuuden takia. Erityisesti koneoppimista hyödyntävien tekniikoiden käyttämistä peliympäristöjen generointiin on vielä alkeellista ja koneoppimisella generoituiduilla korkeuskarttoilla on paljon aihetta jatkotutkimukselle, jotta tekniikkaa voisi hyödyntää oikeissa pelijulkaisuissa.

Asiasanat: proseduraalinen generointi, 3D-malli, polygoniverkko, ympäristö, pelimaailma, kohina-algoritmit

Sisällys

1 Johdanto **1**

2 Tausta **4**

2.1 3D-mallit 4

2.2 Korkeuskartat 5

2.3 Satunnaisuus 6

3 Polygoniverkon generointi **8**

3.1 Tasaisen polygoniverkon generointi 8

3.2 Polygoniverkon muodot 11

3.3 Kohina-algoritmit 12

4 Kehittyneemmät generointitekniikat **17**

4.1 Maapallon topografiadatan käyttö 17

4.2 Syväoppimisen hyödyntäminen 18

4.3 Eroosion simulointi 20

5 Yhteenveto ja pohdinta **22**

Lähdeluettelo **25**

Kuvat

1.1 Tutkimuksen tiedohakuprosessi 3

2.1 Polygoniverkko [2] 5
2.2 Maapallon korkeuskartta [4] 6

3.1 Nelikön generointi Godot-pelimoottorissa 9
3.2 $2^8 \cdot 2^8$ tasanko 11
3.3 Polygoniverkko ja korkeuskartta satunnaisgeneraattorilla 12
3.4 Ympäristö perlin-kohinalla amplitudilla 5 ja taajuudella 0.1 14
3.5 Ympäristö perlin-kohinalla amplitudilla 1 ja taajuudella 0.25 14
3.6 Ympäristö neljän oktaavin fraktaali-kohinalla 15
3.7 Voronoi diagrammi [9] 16

4.1 GAN-mallin luomat korkeuskartat (suomennettu)[10] 19
4.2 Multiskaalaisen eroosiotehostuksen vaiheet (suomennettu) [11] 20

1 Johdanto

Yksi keskeisimmistä elementeistä videopeleissä on ympäristö, joka toimii pohjana pelin sisällölle. Ympäristön pitäisi olla tarpeeksi laaja ja kompleksi, jotta pelaajalla on paljon löydettävää eikä maailma käy tylsäksi. Perinteiset manuaaliset menetelmät kolmiuloitteisten ympäristöjen suunnittelussa vaativat huomattavasti aikaa ja resursseja, mikä rajoittaa pelinkehittäjien kykyä luoda suuria, immersiivisiä pelimaailmoja. Kolmiuloitteisten pelimaailmojen suunnitelu koostuu monesta eri vaiheesta. Maailma tarvitsee topograafiset pintamuodot sekä tekstuurit, ja ympäristöön tulisi asetella kasvillisuutta ja rakennuksia. Tässä tutkielmassa keskitytään kolmiuloitteisten ympäristöjen pintamuotoihin, sillä se toimii pohjana kaikelle muulle.

Proseduraalinen generointi eli sisällön luonti algoritmisesti tarjoaa ratkaisun laajojen ympäristöjen luontiin videopeleissä. Ympäristöjen generointi algoritmisesti mahdollistaa suurien ja uniikkien maailmojen luonnin ilman suurta ajan ja resurssien käyttöä manuaaliseen suunnitteluun. Pelisuunnitelun kannalta ympäristöjen proseduraalinen generaatio tarjoaa myös pelaajille enemmän peliaikaa uniikkien ja laajojen maailmojen avulla.

Ympäristöjä generoivan algoritmin kehittämisessä tulee kiinnittää huomiota muutamaan eri ominaisuuteen. Generoitujen ympäristöjen tulisi olla tarpeeksi laajoja sekä vaihtelevia, jotta pelaajalla on tarpeeksi löydettävää eikä maailma käy tylsäksi. Algoritmin suoritustehokkuus pitää myös optimoida, jotta ympäristö generoituu aiheuttamatta liiallisia latausaikoja, jotka heikentävät pelikokemusta. Ge-

neroidun ympäristön pintamuotojen tulisi muistuttaa mahdollisimman paljon luonnollista eroosiota, sillä tämä lisää immersiota ja realismia maailmaan.

Esimerkki proseduraalista generoiduista ympäristöistä hyödyntävästä pelistä on *No Man's Sky*[1]. Pelin maailma koostuu loputtomasti generoiduista planeetoista, joissa pelaaja voi tutkia ja seikkailla vapaasti. Planeettojen pintamuotojen generaatio perustuu pseudosatunnaisuuteen ja kohina-algoritmeihin. Peli yhdistää proseduraalisesti generoitujen ympäristöjen hyödyt immersiiviseen ja uniikkiin lähes loputtomaan pelikokemukseen.

Tutkielmassa pyritään vastata seuraaviin tutkimuskysymyksiin:

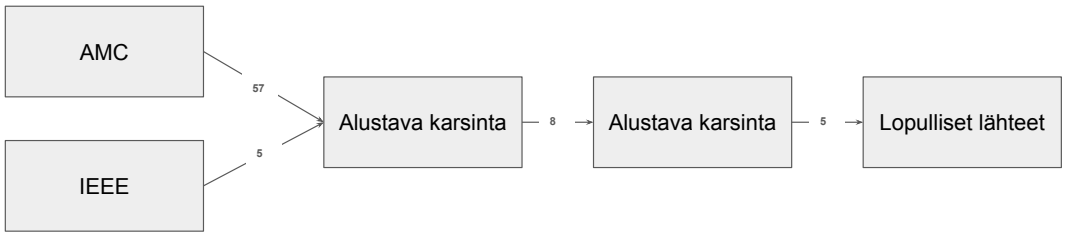
1. Miten kolmiulotteisen ympäristön generointi toteutetaan?
2. Mitä generointimetodeja voi käyttää erilaisten ympäristöjen pintamuotojen luontiin, ja mitkä ovat niiden vahvuudet ja heikkoudet?

Ympäristöjen proseduraalinen generointi on itsessään erittäin laaja aihe, mutta aiheen rajaamiseksi tässä tutkimuksessa keskitytään juuri kolmiuloitteisten ympäristöjen pintamuotojen generointiin. Pintamuodot ovat käytännössä pohja kaikelle muulle ympäristössä, joten tämä on tärkein elementti generoinnista johon tutkimuksen tulisi keskittyä.

Tutkielmassa tiedohaku suoritettiin ACM Digital Library sekä IEEE Xplore -tietopankeissa. Hakulause lopulta usean muutoksen jälkeen päätyi seuraavaan muotoon: *procedural* AND generation* AND game* AND (3D OR three-dimensional) AND (terrain* OR cave* OR landform*) AND (noise* OR "height map*") AND (environment* OR map*)*. Haussa myös rajattiin pois julkaisut, jotka olivat vanhempia kuin viisi vuotta. Alustavassa tiedonhakuvaiheessa oli hieman haasteita muodostaa hyvä hakulause, sillä proseduraalinen generointi on sen verran epämääräinen termi. Haku tuotti paljon julkaisuja, jotka ohittivat aiheen täysin. Otsikoiden ja tiivis-

[1]No Man's Sky -pelin kotisivut: `https://www.nomanssky.com/`

telmien perusteella artikkeleista karsittiin kahdeksan artikkelia. Tarkemman luvun jälkeen tieteellisiä lähteitä jäi viisi.



Kuva 1.1: Tutkimuksen tiedohakuprosessi

Luvussa 2 käydään läpi tutkimuksen ja proseduraalisen generaation kannalta tärkeää tietoa ja termistöä. Seuraavaksi luvussa 3 käsitellään ympäristöä generoivan algoritmin toteuttamista sekä pintamuotojen generoimiseen käytettäviä metodeja. Luvussa 4 käsitellään kehittyneempiä metodeja pintamuotojen parantamiseen. Lopuksi luvussa 5 pohditaan käsiteltyjä metodeja ja mahdollista jatkotutkimusta sekä tehdään yhteenveto tutkielman sisällöstä.

2 Tausta

Kolmiuloitteisten ympäristöjen proseduraalisen generoinnin tutkiminen vaatii tiettyjen perustietojen käsittelemisen. Tässä luvussa käydään läpi kolmiuloitteisten kappaleiden mallintamista digitaalisesti sekä proseduraalisen generaation kannalta tärkeitä termejä.

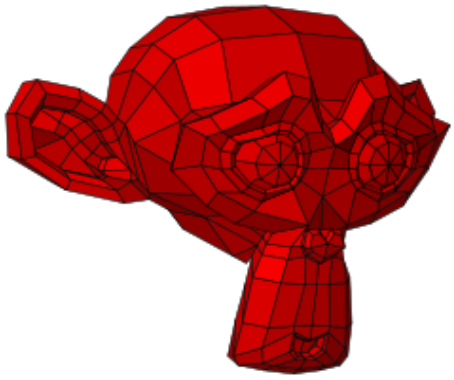
2.1 3D-mallit

Kolmiulotteisia kappaleita mallinnetaan digitaalisesti 3D-malleilla. Polygoniverkko (engl. *polygon mesh*) on geometrinen toteutus luoda kolmiulotteinen avaruus digitaalisesti. Yksinkertaisesti polygoniverkot koostuvat kolmesta eri osasta:

- **Verteksi** (engl. *vertex*), piste avaruudessa, jolla on sijaintikoordinaatit.
- **Särmä** (engl. *edge*), kahden verteksin välinen jana.
- **Tahko** (engl. *face*), kolmen leikkaavan särmän välinen taso.

Yhdistyneitä tahkoja kutsutaan polygoneiksi ja vastaavasti yhdistyneistä polygoneista muodustunut mallia kutsutaan polygoniverkoksi. Polygoniverkoissa tahkot muodostetaan komesta vierekkäisestä särmästä, sillä se on yksinkertaisin tapa luoda monikulmio ja muita kulmioita voi luoda yhdistämällä kolmioita. Usein ohjelmat visuaalisesti mallintaa tahkot nelikulmioina, mutta ohjelma kasittelee näitä tahkoja kahtena vierekkäisenä kolmiona (kuva 2.1). Polygoniverkon verteksit sisältävät myös

UV-koordinaatit, joita käytetään kappaleen visuaaliseen teksturointiin sekä normaalidata, jota käytetään valosäteiden renderöimiseen. Nämä eivät ole tämän tutkielman kannalta keskeisiä, sillä tutkielmassa keskitytään pelkästään polygoniverkon pintamuotoihin.[1]



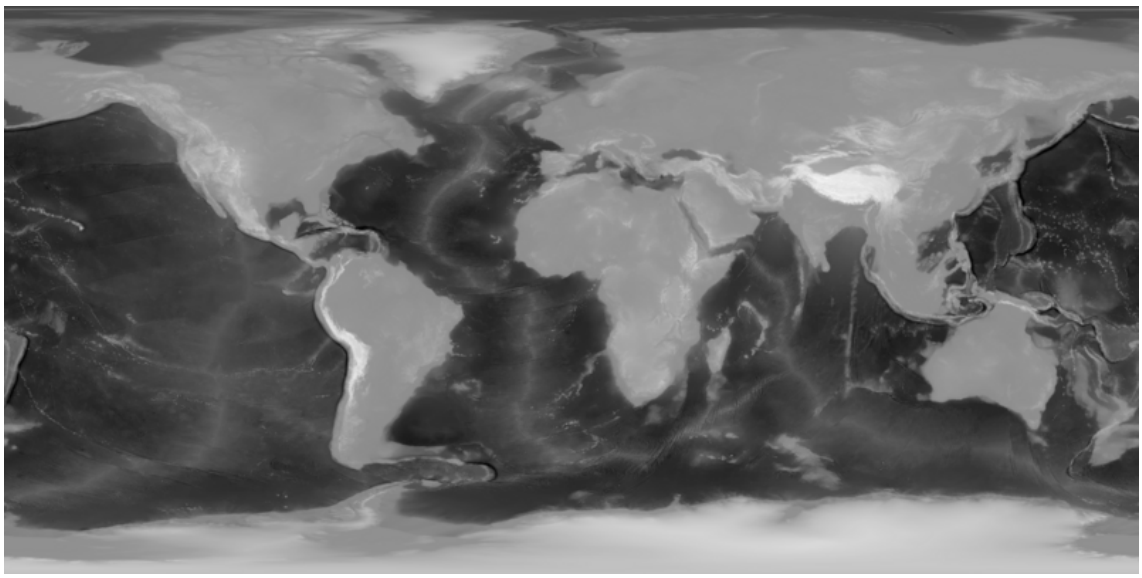
Kuva 2.1: Polygoniverkko [2]

2.2 Korkeuskartat

Korkeuskartat ovat tapa kuvata ympäristön korkeussuhteita kaksiuloitteisessa kuvassa. Kuvassa pikselien väriarvon voi lukea jonakin tiettynä data. Korkeuskarttojen tapauksessa väriarvoon sisällytetty data on kuvan pisteen vastaavan sijainnin korkeusero päätetystä nollapisteestä. Korkeuskartoissa vedenpinta asetetaan tyypillisesti nollapisteeksi, mutta tämä voi periaatteessa olla mikä tahansa päätetty taso. Korkeuskartat tallennetaan yleisesti kuvatiedostoksi, jossa kuvan pikselien värit ovat harmaansävyjä mustan ja valkoisen välillä. Esimerkiksi 8-bittisessä RGB-kuvassa on 256 harmaa-arvoa, nollan ollessa täysin musta ja 256 valkoinen. Kuvassa 2.2 on korkeuskartta maapallosta, jossa merenpohja on nollapiste. Korkeuskartat ovat hyvä tapa visualisoida ympäristön topografiaa. [3]

Proseduraalisen generaation kannalta korkeuskartat on hyödyllisiä, sillä korkeuskartan sisältämät väriarvot voi hyvin suoraviivaisesti kääntää lukuarvoksi. Ympäristön pintatasoa voi peleissä katsoa kaksiuloitteisena ylhäältäpäin ja täten korkeus-

kartta kuvaa suuren osan ympäristön topografiasta. Proseduraalisen generaation kannalta ympärsitön topograafiset korkeusarvot voidaan säilyttää korkeuskartassa, sillä se on korkeusdatan näkökulmasta matriisi tallennettuna tiedostoon.



Kuva 2.2: Maapallon korkeuskartta [4]

2.3 Satunnaisuus

Satunnaisuus on erittäin keskeinen osa generointi algoritmin luomista. Satunnaislukugeneraattori (engl. *random number generator*, RNG) on algoritmi, jolla luodaan satunnaisia liukulukuja 0 ja 1 välillä. Käytännössä satunnaislukugeneraattorit eivät saavuta todellista satunnaisuutta ja näiden algoritmien tulosta kutsutaankin usein pseudosatunnaisuudeksi [3]. Satunnaisgeneraattorin luoman satunnaisuuden laatuun tai kompleksisuuden ei syvennytä tässä tutkielmassa. Tutkielmassa oletetaan, että on käytössä satunnaislukugeneraattori, jonka palauttamiin arvoihin muun algoritmin satunnaisuus osittain perustuu.

Pseudosatunnaisuudessa on hyvänä etuna se, että algoritmin tuloksia voidaan toistaa siementen (engl. *seed*) avulla. Satunnaislukugeneraattorille voi syöttää tietyn

siemenen ja algoritmi palauttaa aina samat luvut samassa järjestyksessä. Tämä on hyödyllistä erityisesti proseduraalisessa generoinnissa, sillä siementen avulla voidaan luoda sama ympäristö uudestaan.

3 Polygoniverkon generointi

Tässä luvussa käsitellään yksinkertaisen generaatioalgoritmin luontia. Generaatioalgoritmit luodaan Godot-pelimoottorissa visuaalisina esimerkkeinä. Pelimoottorin 3D-avaruuden ja renderöinnin toteukseen ei syvennytä tässä tutkielmassa. Pelimoottoreissa saatavilla olevat polygoniverkkojen luontiin tarkoitetut työkalut toimivat yleisesti samalla idealla[1]. Algoritmin toiminta itsessään esitetään ohjelmalistuaksessa pseudokoodina.

3.1 Tasaisen polygoniverkon generointi

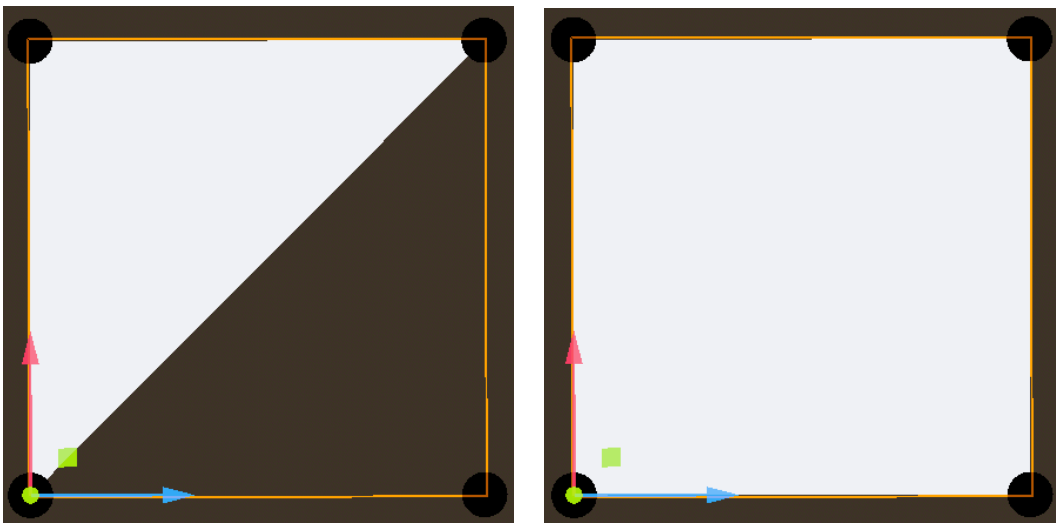
Polygoniverkon algoritmisessa generoinnissa ensimmäisenä askeleena on saada luotua neliön muotoinen polygoniverkko, joka toimii rakennuspalana muulle ympäristölle. Neliön muotoisen polygoniverkon eli nelikön voi luoda kahdella vierekkäisellä kolmiotahkolla. Nelikön luontiin tarvitaan yhteensä neljä verteksiä, sillä nelikön sisäiset kolmiot voivat käytää samoja verteksejä näitä yhdistävällä suoralla. Verteksit tarvitsevat sijainnin kolmiuloitteisessa avaruudessa, joka on pelimoottoreissa yleisesti Vector3 tyypitetty, joka on käytännössä kolmen liukuluvun sisältävä tietue (engl. *struct*). Haluamme luodun verkon korrelloivan korkeuskartan kanssa myöhemmin, joten verkon tulisi olla avaruuden korkeusakselilla täysin tasainen. Verteksien sijainnin y-akselilla voi täten asettaa nollaan.

[1]Ohjeita polygonien luontiin Godot-moottorissa löytyy Godotin dokumentaatio sivuilta: `https://docs.godotengine.org/en/stable/tutorials/3d/procedural_geometry/arraymesh.html`

Nelikön luonnissa täytyy myös määritellä verteksien välille tarvittavat särmät. Tämä tapahtuu lisäämällä luotujen verteksien indeksit halutussa järjestyksessä indeksilistaan. Käytetyn pelimoottorin renreröintiputki luo vierekkäisten indeksien välille särmän ohjelman suoritusvaiheessa. Annettujen indeksien järjestys voi vaihdella pelimoottorien välillä; esimerkiksi Godot-moottorissa annetaan indeksit myötäpäivästi. Ohjelmalistauksessa 1 kuvataan nelikön generointi ja kuvassa 3.1 nähdään nelikön muodostuminen kahdesta kolmiosta Godot-pelimoottorissa. [3]

Ohjelmalistaus 1 Nelikön generointi

```
function generate_quad():
    vertices = [
        Vector3(0, 0, 0),
        Vector3(1, 0, 0),
        Vector3(0, 0, 1),
        Vector3(1, 0, 1)]
    indices = [
        0, 1, 3,  #ensimmäinen kolmio
        0, 3, 2   #toinen kolmio
    ]
    quad_mesh = Mesh()
    quad_mesh.vetices = vertices
    quad_mesh.indices = indices
```



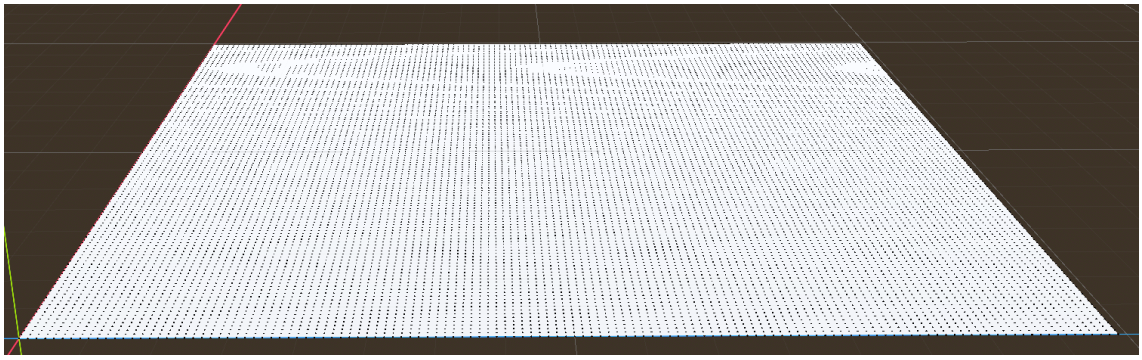
(a) (b)

Kuva 3.1: Nelikön generointi Godot-pelimoottorissa

Nelikkö ei itsestään käy polygoniverkoksi, ellei haluttu ympäristö ole täysin tasainen. Jotta ympäristöön voi luoda muotoja, täytyy nelikkö jakaa osiin eli tavoitteena on luoda ruudukko yhdistyneitä neliköitä. Algoritmin laajentaminen voidaan toteuttaa iteroimalla neliköiden luontia x- ja z-akseleilla. Ympäristön verteksien määrä vaikuttaa paljonko yksityiskohtaista deformaatiota polygoniverkkoon voi myöhemmin luoda. Haluamme luoda ympäristön, jossa on mahdollista luoda korkeuskarttaan perustuvaa topografiaa, joten tarvitsemme melko paljon verteksejä. Verteksien määrän kasvaessa eksponentiaalisesti täytyy ottaa huomioon ohjelman suoritustehokkuus, sillä verteksien suuri määrä on erittäin raskasta pelimoottorin renderöintiputkelle suorittaa. Tutkimuksesta varten luodaan tasanko, jossa on verteksejä $2^8 \cdot 2^8$. Ohjelmalistauksessa 2 nähdään päivitetty generointialgoritmi ja kuvassa 3.2 ohjelman tuottama polygoniverkko.

Ohjelmalistaus 2 Tasangon generointi

```
function generate_plane():
    size=128
    vertices = []
    indices = []
    for z in range(size):
        for x in range(size):
            vertices.append(Vector3(x,0,y))
            if x>0 and z>0:
                indices = [
                    #ensimmäinen kolmio
                    (z-1)*size+(x-1), (z-1)*size+x, z*size+x,
                    #toinen kolmio
                    (z-1)*size+(x-1), z*size+x, z*size+(x-1)
                ]
    quad_mesh = Mesh()
    quad_mesh.vetices = vertices
    quad_mesh.indices = indices
```



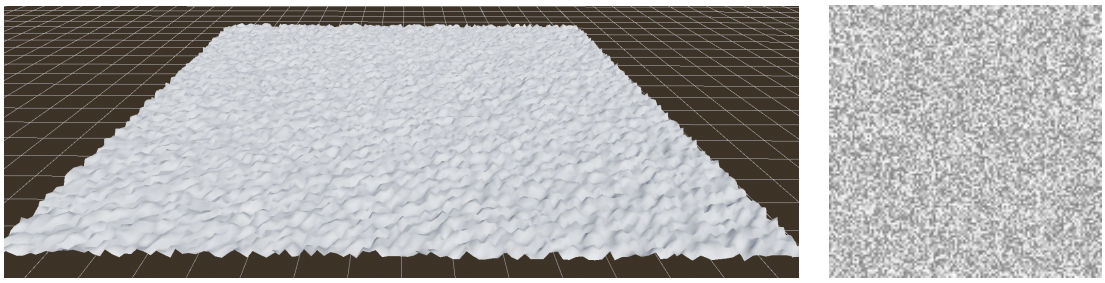
Kuva 3.2: $2^8 \cdot 2^8$ tasanko

3.2 Polygoniverkon muodot

Topograafisten muotojen lisäämistä varten tulisi ensin generoida korkeuskartta. Algoritmiin on sisällytetty jo polygoniverkon verteksien korkeusarvo, mutta tällä hetkellä algoritmi asettaa jokaisen verteksin y-koordinaatin arvoksi nollan. Vaikka kaikilla vertekseillä on täysin sama korkeusarvo voisi tälle ympäristölle silti luoda korkeuskartan. Haluamme korkeuskartan pikselien korreloivan polygoniverkon verteksien kanssa, joten korkeuskartan resoluutio tulisi olla $2^8 \cdot 2^8$. Tällä hetkellä korkeuskartan korkeusarvojen skaalaus on vakio eli harmaasävyisessä korkeuskartassa se olisi täysin musta, sillä jokainen arvo on nolla.

Yksinkertaisin tapa generoida muutosta korkeuskartan arvoihin on käyttää luvussa 2.3 esitettyä satunnaislukugeneraattoria. Pelimoottoreissa on yleisesti saatavilla satunnaisgeneraattori, jota voi käyttää vain kutsumalla. Generointialgoritmiin tämän voi toteuttaa helposti lisäämällä vertkeisien y-koordinaatin sijainniksi satunnaisgeneraattorin kutsun. Täten jokainen verteksi saa y-koordinaatin arvoksi satunnaisgeneraattorin palautuksen, joka on satunnainen luku. Korkeuskartan luonnissa täytyy vastaavasti iteroida pysty- ja vaaka-akseleilla jokaisen pikselin läpi ja asettaa satunnaisarvoa vastaava RGB-arvo. Tämän voi hyvin toteuttaa samassa silmukassa, jossa luodaan polygoniverkko. Kun korkeuskartta luodaan samassa silmukassa polygoniverkon kanssa, voi satunnaisarvon suoraan sijoittaa verteksin sijainniksi.

Halutessa korkeuskartan arvojen skaalausta polygoniverkon sijaintiin voi muuttaa lisäämällä amplitudivakion. Vakion voi yksinkertaisesti kertoa korkeusarvon kanssa, kun se sijoitetaan sijaintiin. Päivitetyn ohjelman tulostama polygoniverkko sekä korkeuskartta näkyy kuvassa 3.3.



(a) polygoniverkko (b) korkeuskartta

Kuva 3.3: Polygoniverkko ja korkeuskartta satunnaisgeneraattorilla

Ohjelman tulostamassa polygoniverkossa on haluttuja topograafisia muutoksia, mutta tässä metodissa ilmenee paljon ongelmia. Ympäristö vaikuttaa erittäin rosoiselta, sillä vierekkäisten verteksien korkeusarvoilla ei ole mitään graduaalista korrellaatiota. Kuvassa 3.3 nähtävässä tulostetussa polygoniverkossa käytettiin amplitudina arvoa kaksi, mutta amplitudin muuttamisella ei vielä paranna tulosta juurikaan, sillä satunnaisgeneraattorin käyttö ei pohjimmiltaan tuota luonnollisia tuloksia. Pelkän satunnaislukugeneraattorin käytössä ei ole käytännössä mitään kontrollia ja tulos ei ole kelvollinen lähes missään tilanteessa. [3]

3.3 Kohina-algoritmit

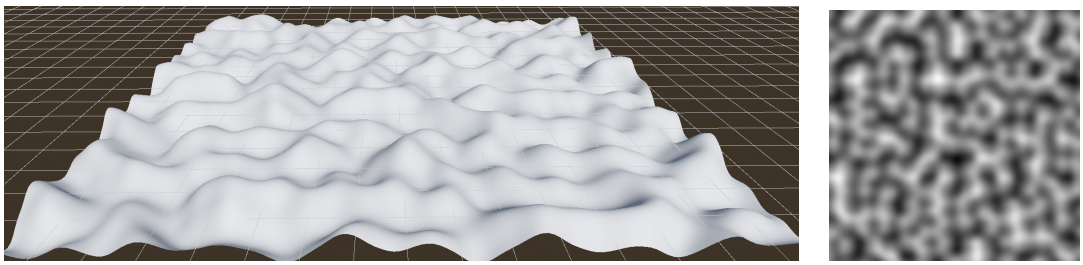
Proseduraalisesti generoiduissa ympäristöjen halutaan olevan uniikkeja, mutta satunnaisten muotosten tulisi vaikuttaa mahdollisimman realistiselta topograafiselta eroosiolta. Kohina-algoritmit (engl. *noise algorithms*) ovat hyödyllisiä työkaluja erityisesti proseduraalisessa generoinnissa, sillä niillä pystyy toteuttamaan pseudosatunnaisuutta säilyttäen enemmän kontrollia. Tarkemmin kohina-algoritmeista gra-

dienttikohinat ovat hyödyllisiä, sillä näissä toteutuksissa vierekkäisten datapisteiden muuttuu graduaalisesti. Algoritmit voidaan toteutta missä tahansa ulottuvuudessa, mutta tyypillisesti ne tunnetaan kaksiuloitteisena toteutuksena. Kaksiuloitteisena algoritmia voidaan hyödyntää juuri korkeuskarttojen generointiin. [5]

Kohina-algoritmeista tunnetuin on Kevin Perlinin luoma Perlin-kohina (engl. *Perlin noise*). Perlin-kohina alunperin kehitettiin luomaan luonnollisempia proseduraalisia tekstuureja. Perlin-kohinan tomintaperiaatteena on jakaa datapisteet ruudukkoon hiloja (engl. *lattice*). Jokaiselle hilalle annettaan satunnainen gradienttivektori. Kohinan laskeminen tapahtuu selvittämällä ensin hilan kulmapisteet. Tämän jälkeen kulmapisteille annetaan omat gradienttivektorit sekä lasketaan siirtymävektori. Gradienttien vaikutuksen saa täten gradienttivektorien ja siirtymävektorien pistetulolla. Lopullisen kohinan saa interpoloimalla kulmapisteiden gradienttivaikutukset. [5] Kevin Perlin loi Perlin-kohinasta parannetun implementaation Simplex-kohinan. Simplex-kohina on tehokkaampi ja tuottaa vähemmän artifakteja, joten nykyään tätä algoritmia käytetään usein alkuperäisen sijasta. [6]

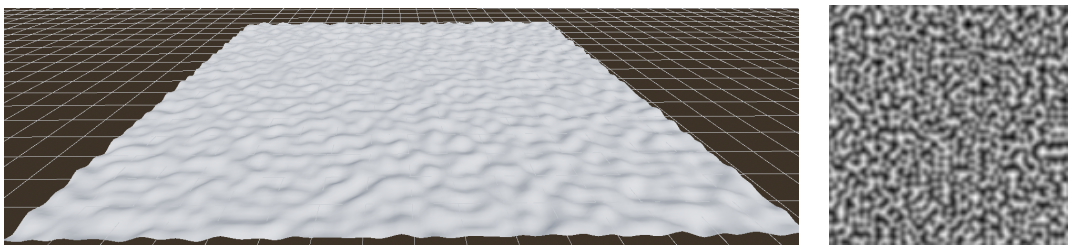
Perlin-kohinan on hyvä ratkaisu ympäristön korkeuskartan parantamiseen, sillä vierekkäisten pisteiden korrellaation puute ei ole ongelma gradienttikohinassa. Perlin-kohinaa käyttäessä saadaan myös paljon enemmän kontrollia, sillä kohinan amplitudia ja taajuutta voi säädellä. Perlin-kohinassa taajuus merkitsee, sitä kuinka tiheästi kohinaa generoidaan eli suurempi taajuus vastaisi jyrkempiä muutoksia vierekkäisten pisteiden välillä. [3] Tämän lisäksi kohina-algoritmiin voi asettaa siirrosarvon eli kuinka paljon tulosta siirretään esimerkiksi x- ja y-akseleilla. Perlin- tai Simplex kohina-algoritmien toteutuksiin ei syvennytä tämän enempää, mutta lähes kaikissa pelimoottoreissa on jokin kohina-algoritmin toteutus saatavilla.

Päivitetty ohjelma suoritettiin kaksi kertaa eri syötetyillä amplitudi- ja taajuusarvoilla, joiden tulokset nähdään kuvissa 3.4 ja 3.5. Tuloksissa näkyy selvästi suuri parannus pelkän satunnaisgeneraattorin käyttöön verrattuna.



(a) polygoniverkko (b) korkeuskartta

Kuva 3.4: Ympäristö perlin-kohinalla amplitudilla 5 ja taajuudella 0.1



(a) polygoniverkko (b) korkeuskartta

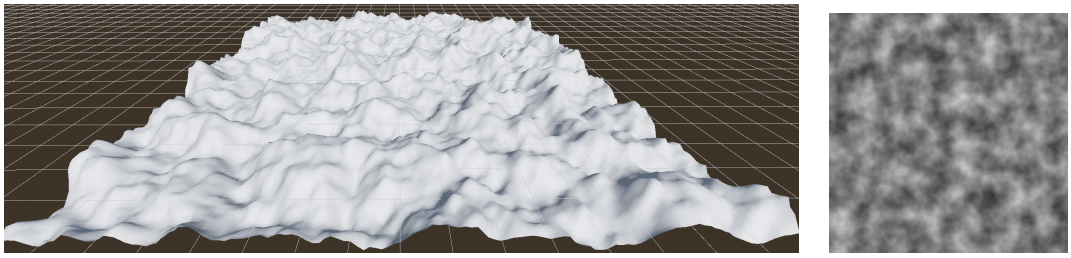
Kuva 3.5: Ympäristö perlin-kohinalla amplitudilla 1 ja taajuudella 0.25

Ympäristö vaikuttaa paljon luonnollisemmalta, kun korkeuserot korrelloivat graduaalisesti vierekkäisten verteksien välillä. Tuloksista huomataan myös kontrollin nousu, sillä amplitudia ja taajuutta muuttamalla pystyy vaikuttamaan huomattavasti ympärsitön muotojen jyrkkyykseen ja tiheyteen. Kuvassa 3.4 nähdään vuoristomainen ympäristö ja kuvassa 3.5 nähdään karu tasanko. Säätämällä aplitudi- ja taajuus arvoja on mahdollista saavuttaa esimerkiksi jyrkempi vuoristo tai vielä tasaisempi tasanko.

Perlin-kohinan käytössä kuitenkin huomataan, että ympärsitö vaikuttaa erittäin sileältä. Erityisesti kuvassa 3.4 nähdään vuoristoinen maisema ja pohjimmaiset muodot näkyy hyvin, mutta oikeassa vuoristossa näiden suurten vuorten lisäksi paljon pieniä eroosion aiheuttamaa epätasaisuutta.

Fraktaalikohina (engl. *fractal noise*) on yksi ratkaisu tähän ongelmaan. Fraktaalikohina ei ole erillinen kohina-algoritmi vaan tapa yhdistää useampi kohina-algoritmin tulos yhteen parempaan tulokseen. Fraktaalikohina toimii siten, että

kohina-algoritmi suoritetaan useaan kertaan. Seuraavaan suoritusiteraation siirtyessä kohina-algoritmin tiheys tuplataan, mutta kohinan amplitudi puolittuu alkuperäisestä. Seuraava iteraatio tämän jälkeen päällystetään peittokuvana (engl. *overlay*) alhaisemmalla vahvuudella eli opasiteetilla (engl. *opacity*). Fraktaalikohinan ”iteraatiotasoja” kutsutaan oktaaveiksi ja oktaaveja voi teoriassa olla loputtomasti. Iteraation opasiteetin tai taajuuden muutoksen ei tarvitse olla aina puolet ja kaksi, mutta nämä antavat usein tarpeeksi hyvän tuloksen. Kuvassa 3.6 nähdään päivitetyn algoritmin tulos. [3][7][8]



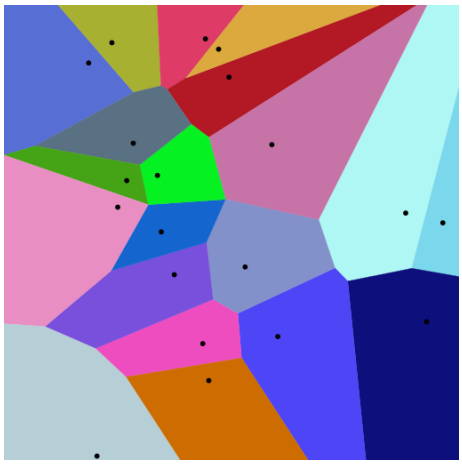
(a) polygoniverkko (b) korkeuskartta

Kuva 3.6: Ympäristö neljän oktaavin fraktaali-kohinalla

Kuvasta 3.6 huomataan, että fraktaalikohinaa käyttämällä päästään eroon ympäristön sileästä vaikutelmasta. Ensimmäinen oktaavi määrittelee ympäristön suuret pohjimmaiset muodot ja seuraavat oktaavikerrokset lisäävät pienempiä pintamuutoksia, jotka saavat ympäristön muistuttavan luonnollisempaa eroosiota. Fraktaalikohinan tuomat uudet arvot oktaaviarvot sekä myös amplitudi- ja taajuus arvot antavat fraktaalikohinan tulokselle paljon mahdollisuutta mukauttaa se haluttuun ympäristötyyppiin. Oli ympäristö tasaisempi tasanko tai jyrkkä vuoristo, monen tyyppiset ympäristöt ovat mahdollista luoda fraktaalikohinalla ja ne näyttävät yksityiskohtaisimmilta käyttämällä suurempia oktaaveja.

Fraktaalikohinassa ilmenee kuitenkin ongelmana se, että kohinan vuorten huiput toistuvat eikä tällä metodilla ole mahdollista generoida yksittäisiä vuoria ja kanjoneita tasaisen maan lisäksi. Voronoi-tesselointi (engl. *Voronoi tessellation*) on matemaattinen menetelmä, jolla voidaan jakaa alueita osiin pistejoukkojen perus-

teella. Esimerkki Voronoin muodostamista muodostamista monikulmioista nähdään kuvassa 3.7.



Kuva 3.7: Voronoi diagrammi [9]

Jokainen osa eli kenno koostuu alueesta, joka on lähempänä tiettyä pistettä kuin mitään muuta pistettä joukossa. Tämän jaon avulla voidaan määrittää haluttujen vuorten sijainti ympäristössä. Generointi algoritmi voisi siis arpoa tiettyjen voronoi pisteiden kohdalle tietyn vuoren korkeusarvon. Jos tämän tiedon yhdistää muuhun polygoniverkon generointi algoritmiin, voidaan kyseessäolevan voronoi pisteen korkeusarvoa kasvattaa korkeusarvolla. Korkeusarvoon voidaan lisätä graduaalisuutta vähentämällä korkeusarvoa kyseisen voronoikennon pisteen etäisyydestä keskipisteestä. Täten generoitu vuori saa selkeän huipun.

4 Kehittyneemmät generointitekniikat

Tähän mennessä tutkielmassa olemme käsitelleet kohina-algoritmeihin perustuvia generointitekniikoita. Kohina-algoritmit ovat nopeita ja niillä on monipuolisia aplikaatioita. Tässä luvussa käsitellään kehittyneempiä tekniikoita luonnollisen ympäristön generoinnin saavuttamiseksi.

4.1 Maapallon topografiadatan käyttö

Luvussa 3.3 käsiteltiin fraktaalikohinaa, joka paransi generoidun ympäristön yksityiskohtia ja realismia pelkkään Perlin kohinaan verrattuna. Fraktaalikohinan tuottamilla ympäristöillä yritetään emuloida luonnollista eroosiota, mutta yksityiskohdat eivät oikeasti vastaa oikeaa hierarkiaa, sillä se perustuu satunnaisten gradienttien yhdistämiseen.

Generaatio algoritmien tavoitteena on jäljitellä oikeaan luontoon muodostuneen eroosion luomaa topografiaa. Loogista olisi siis käyttää maapallosta kerättyä topografiadataa. Maapallosta kerättyä topografiadataa on saatavilla jonkin verran ja sitä on kerätty eri tavoin. Topologiadataa on saatavilla esimerkiksi Nasan SRTM(engl. *Shuttle Radar Topography Mission*) elevaatiokartan tai Google Earthin kautta [6][10]. Topografiadatan raa'an elevaatiodatan voi kääntää harmaansävyiseksi korkeuskartaksi. Kun korkeuskartan resoluutio ja haluttu skaalaus raa'asta datas-

ta korkeuskarttaan, voidaan topografiadataa iteroida jokaiseen korkeuskartan pikseliin. Luodut korkeuskartat eivät ole uniikkeja, sillä ne käännettiin topologiadatasta, mutta korkeuskarttoja voidaan hyödyntää datasettinä.

4.2 Syväoppimisen hyödyntäminen

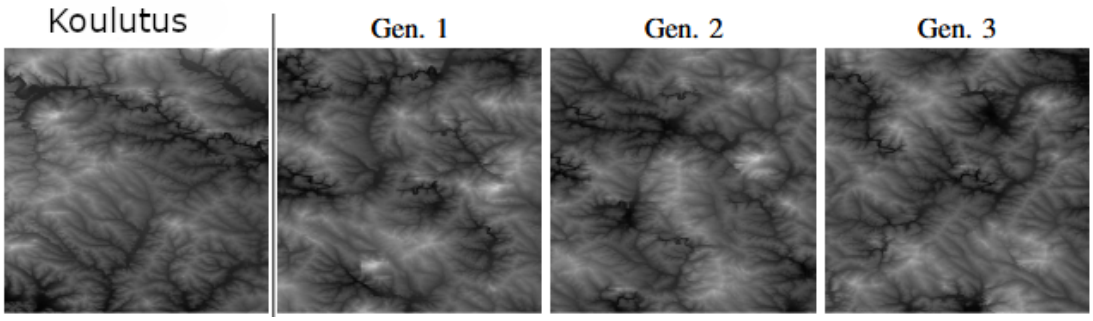
Tekoäly ja koneoppimisen eri muodot ovat olleet nousussa viime aikoina. Syväoppiminen on koneoppimisen muoto, jossa dataa käsitellään eri kerroksissa neuroverkkoa. Ohjelma pyrkii oppimaan annetun datan piirteitä ensin matalan tason ominaisuuksia siirtyen myöhemmissä kerroksissa korkeamman tason yksityiskohtaisempiin ominaisuuksiin. Sisältöä tuottavat syväoppimista hyödyntävät applikaatiot ovat kehittyneet hyvin nopeasti. Syväoppimisrakenteen voi hyvin sisällyttää ympäristoä proseduraalisesti generoivaan algoritmiin.

Generatiivinen kilpaileva verkosto (engl. *generative adversarial network*, GAN) on koneoppimisrakenne, joka hyödyntää kahta syvää neuroverkostoa generaattori ja diskriminaattori. Generaattorin tavoitteena on tuottaa synteettistä dataa muistuttaen mahdollisimman paljon annettua datasettiä. Diskriminaattorin tavoite on erottaa generaattorin luoma synteettinen datapisteet alkuperäisen datasetin datapisteistä. GAN-rakenteen koulutus perustuu molempien verkkojen samanaikainen oppiminen, jossa generaattori kehittyy datasettiä muistuttavan synteettisen datan tuottamisessa diskriminaattorin arvion avulla ja diskriminaattori kehittyy oikeiden ja synteettisten datapisteiden erottamisessa. GAN-rakenteen tavoitteena on kartoittaa syötesetin ominaisuuksia, jota voi käyttää luomaan alkuperäistä syötesettiä jäljittelevän tuloksen.

GAN-mallin kouluttaminen korkeuskarttojen luomiseen vaatii datasetin keräämisen. Luvussa 4.1 mainituilla tavoilla voidaan valita halutun ympäristön kaltainen alue koulutusmalliksi, josta sitten luodaan korkeuskartta. Tarkoituksena on luoda korkeuskartasta tarpeeksi iso, jotta sen voi jakaa useaan identtisen kokoisiin sektioi-

hin. Jokaisen koulutusiteraation alussa koulutusdatasta valitaan satunnainen sektio, jota diskriminaattori sitten vertaa generaattorin luomaan tulokseen. GAN-rakenteen suurin haaste on generaattorin ja diskriminaattorin kehittymisen tasapainon ylläpitäminen, sillä diskriminaattori usein kehittyy nopeammin kuin generaattori [6][10].

Spickin, Cowlingin ja Walkerin julkaisussa [10] suoritettiin GAN-malliin perustuva proseduraalisten korkeuskarttojen koulutus. Koulutus käytti luvussa 4.1 mainitusta SRTM elevaatiokarttasta 100km^2 aluetta, joka muunnettiin 0.5 suuruisella refaktoroinnilla korkeuskartaksi. Korkeuskartta jaettiin 32 identtisen kokoisiksi paloiksi ja koulutuksiteraation alussa näistä valittiin aina satunnaisesti yksi. Koulutuksen aikana todettiin, että diskriminaattorin pitäisi kouluttaa vain puolessa iteraatiosykleistä, mutta generaattoria kolutettiin jokaisella. Koulutusta suoritettiin noin 4 tuntia useaan kertaan. Koulutuksen jälkeen malli tuotti $2400 \cdot 2400$ resoluution korkeuskarttoja jopa 50 millisekunnissa GTX 1080Ti grafiikkakortilla (yksi tutkimuksen generoiduista tuloksista näkyy kuvassa 4.1). [10]



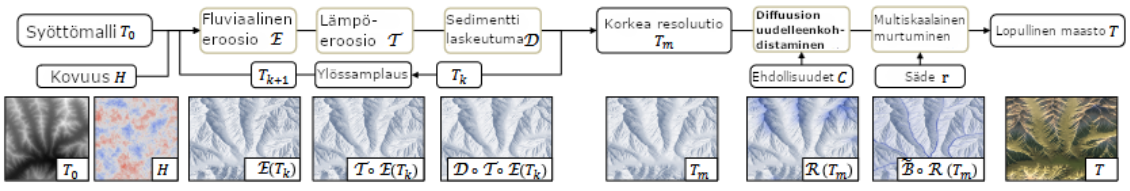
Kuva 4.1: GAN-mallin luomat korkeuskartat (suomennettu)[10]

Koulutetun GAN-mallin tulokset ovat huomattavasti parempia kohinaalgoriteihin verrattuna muutamasta syystä. GAN-malli jäljittelee koulutusympäristöön nähden luonnollisen hyvin ja yksityiskohdat ovat huomattavasti parempia kuin fraktaalikohinaa hyödyntäen. Perlin-kohinalla on tapa toistaa samoja ominaisuuksia, mutta GAN-malliin mahdollisuus luoda suuren ja yksityiskohtaisen koulutusmallin perusteella vähentää ominaisuuksien toistuvuutta. Malliin perustuvan gene-

raation suuri ongelma on kuitenkin se, että suurten maailmojen luonti on vaikeaa sillä, muiden generoitujen korkeuskarttojen toisiinsa yhdistäminen on käytännössä mahdotonta tällä mallilla. [6][10]

4.3 Eroosion simulointi

Generoitua ympäristöä voidaan parantaa maaston vahvistuksella(engl. *terrain amplification*). Maaston vahvistus tarkoittaa siis sitä, että otetaan jo olemassa oleva korkeuskartta syötteeksi ja sen piirteitä vahvistetaan algoritmisesti. Schottin, Galinin, Guérinin ja muiden artikkelissa [11] käsitellään maaston tehostamisesta multiskaalaisella eroosioalgoritmilla. Algoritmi toimii siten, että syötteenä annettua alhaisemman resoluution korkeuskarttaa käsitellään eri eroosiotehostus vaiheissa (vaiheet näkuvat kuvassa 4.2). Algoritmissa simuloidaan veden virtauksen muodostamaa eroosioita, lämpöeroosiota sekä sedimenttien laskeutumaa. Algoritmi iteroivasti kasvattaa korkeuskartan resoluutiota eroosiosimulaation heikkenevällä voimakkuudella jokaisella iteraatiokerralla samalla periaatteella kuin fraktaalikohinassa. Multiskaalaisen eroosiosimulaation jälkeen mahdollisia virheellisiä korkeushuippuja koitetaan parantaa tasoituksen uudelleenkohdistamisella(engl. *diffusion retargeting*) sekä multiskaalainen rikkomusalgoritmi(engl. *multi-scale breaching*), joka auttaa luomaan jokiin ja kanjoneihin johdonmukaiset kuivatusalueet. [11]



Kuva 4.2: Multiskaalaisen eroosiotehostuksen vaiheet (suomennettu) [11]

Maaston vahvistuksen tuloksia voidaan säädellä muuttamalla algoritmiin sisällettyä kovuus syötettä, joka vaikuttaa eroosiosimulaation vahvuuteen. Lisäksi algo-

ritmin laatua ja aikakompleksisuutta määrittelee paljolti tavoiteltu resoluutio. Tuloksen resoluution kasvaessa algoritmin aikakompleksisuus kasvaa eksponentiaalisesti. Tutkielmassa päädyttiin siihen tulokseen, että uusimmilla grafiikkakorteilla 8192 pikselin resoluutioon päästiin parinkymmenen millisekunnin suoritusajalla. Suoritusaika on varsin hyvä, mutta tässä täytyy ottaa myös huomioon se, että generoidut ympäristöt ovat todella laajoja ja tämä on todellisessa toteutuksessa vain pieni sektio kokonaisesta ympäristöstä. [11]

5 Yhteenveto ja pohdinta

Tutkielman tavoitteena oli tutkia kolmiuloitteista videopeliympäristön proseduraalista generointialgoritmin toteuttamista ja eri metodija proseduraalisen generaation toteuttamiseen. Valitut lähteet käsittelivät aihetta eri näkökulmista.

Tutkimuksen tiedonhakuvaiheessa huomasin, että tieteellisiä tutkimuksia aiheesta oli hankala löytää. Proseduraalinen generaatio on aiheena sen verran laaja, että monet tieteelliset julkaisut sivusivat aihetta jollakin tavalla, mutta eivät kuitenkaan vastannut haluttuja tutkimuskysymyksiä. Monet näistä artikkeleista käsittelivät esimerkiksi kaksiuloitteisten pelien maailmoja tai ympäristöjen rekonstruktiota polygoniverkoiksi. Hakulausetta muuntamalla löytyi lopulta muutama hyvä julkaisu vastaamaan tutkimuskysymyksiä.

Tieteellisissä julkaisuissa ilmeni yhtenevä piirre, jossa monet proseduraaliseen generaatioon liittyvät peruspilarit ohitettiin tutkimuksissa aika nopeasti ja mainittiin julkaisuksien taustakappaleissa vain muutamalla virkkellä. Näillä tarkoitan juuri esimerkiksi kohina-algoritmeilla toteutettua proseduraalista generaatiota. Kohina-algoritmien ja erityisesti Perlin-kohinan käyttö on videopelien ympäristöjen luonnissa erittäin oleellinen osa, sillä monet modernitkin pelit(esimerkiksi No Man's Sky) perustavat niiden ympäristöjen luonnin lähinnä näihin algoritmeihin. Tästä huolimatta kohina-algoritmeihin perustuvasta proseduraalisesta generaatiosta oli hankala löytää hyviä tieteellisiä lähteitä.

Tieteellisten lähteiden vähäisestä määrästä huolimatta tutkimuksen kysymyk-

siin saatin melko hyviä vastauksia. Ensimmäinen kysymys *Miten kolmiulotteisen ympäristön generointi toteutetaan?* saa luvussa 3 yksiköhtaisesti alkuvaihesta monimutkaisempiin tapoihin vastauksen. Toinen kysymys *Mitä generointimetodeja voi käyttää erilaisten ympäristöjen pintamuotojen luontiin, ja mitkä ovat niiden vahvuudet ja heikkoudet?* saa vastauksen luvussa 3 käytyjen tekniikoihin esittelyn jälkeen ja seuraava esitelty tekniikka pyrkii korjaamaan edellisen haittoja. Tässä luvussa lähinnä keskityttiin kohina-algoritmeihin ja tämä lähinnä johtui siitä, että lähteissä ei käsitelty juuri muita tekniikoita tämän lisäksi. Toki luvun lopussa käsitelty voronoi-tessalointi oli hyvä lisäys pelkkiin kohina-algoritmeihin perustuvaan generointiin. Tästä huolimatta kohina-algoritmeista saatiin melko kattava kuva ja niiden eri hyödyistä ja haitoista, vaikka ehkä olisi voinut syventyä enemmän myös muihin tekniikoihin.

Luvussa 4 käsiteltiin kehittyneempiä tekniikoita proseduraalisen generaation kannalta, joiden tavoitteena on parantaa tuotettujen ympäristöjen realismia sekä vähentää kohina-algoritmien käytössä ilmenevää ympäristön ominaisuusten toistuvuutta. Erityisesti GAN-mallin käyttäminen korkeuskarttojen generoinnissa selvästi parantaa tulosten realismia. Koneoppimisrakenteiden käytössä kuitenkin ilmenee se ongelma, että proseduraalisesti tuotettuja korkeuskarttoja on tämänhetkisillä käytänteillä mahdotonta yhdistää toisiinsa. Tämän takia suurten maailmojen luonti on vielä hyvin hankalaa, sillä koko ympäristön tulisi koostua yhdestä korkeuskartasta.

Kohina-algoritmit ja erityisesti Perlin-kohina ovatkin vielä käytössä tästä syystä moderneissa peleissä. Koneoppimista hyödyntävään ympäristöjen generointiin on vielä paljon tilaa jatkotutkimukselle. Koneoppimisella generoitujen korkeuskarttojen kyvyttömyys liittyä toisiinsa saumattomasti on vielä niin suuri este, että teknologiaa ei ole vielä voinut realistisesti hyödyntää peleissä. Tulevaisuudessa aihetta voisisi tutkia enemmän ja löytää jonkinlainen ratkaisu generoitujen palasten yhdistämiseen, sillä tutkimuksessa ilmeni koneoppimisen mahdollistaa suuren askeleen ympäristöjen

realismissa. Myös korkeuskarttojen parantamista eroosion simuloinnilla mahdollistaa suurempien resoluutioiden ympäristöt luonnin, jos ohjelman proseduraalisesti luoma korkeuskartta on alhaisempaa resoluutiota. Maaston vahvistustekniikat ovat vielä hyvin vähän käytetty proseduraalisessa generoinnissa ja näillekkin tekniikoille on varmasti paljon mahdollisuuksia jatkotutkimuksille.

Lähdeluettelo

[1] S. Tiigimägi. "Introduction to 3D Polygon Mesh". (2022), url: https://3dstudio.co/polygon-mesh/ (viitattu 09. 09. 2024).

[2] Inductiveload, Wikimedia Commons. (2008), url: https://commons.wikimedia.org/wiki/File:Suzanne.svg (viitattu 28. 01. 2025).

[3] K. S. Emmanuel, C. Mathuram, A. R. Priyadarshi, R. A. George ja J. Anitha, "A Beginners Guide to Procedural Terrain Modelling Techniques", teoksessa *2019 2nd International Conference on Signal Processing and Communication (ICSPC)*, 2019, s. 212–217. DOI: 10.1109/ICSPC46172.2019.8976682.

[4] Avsa, Wikimedia Commons. (2023), url: https://commons.wikimedia.org/wiki/File:World_elevation_map.png (viitattu 28. 01. 2025).

[5] A. Biagioli. "Understanding Perlin Noise". (2014), url: https://adrianb.io/2014/08/09/perlinnoise.html (viitattu 18. 11. 2024).

[6] R. Spick ja J. Walker, "Realistic and Textured Terrain Generation using GANs", teoksessa *Proceedings of the 16th ACM SIGGRAPH European Conference on Visual Media Production*, sarja CVMP '19, London, United Kingdom: Association for Computing Machinery, 2019, ISBN: 9781450370035. DOI: 10.1145/3359998.3369407.

[7] N. Blevins. "Fractal Noise". (2020), url: http://neilblevins.com/art_lessons/fractal_noise/fractal_noise.html (viitattu 18. 11. 2024).

[8] K. Satyadama, R. F. Rachmadi ja S. M. S. Nugroho, ”Procedural Environment Generation for Cave 3D Model Using OpenSimplex Noise and Marching Cube”, teoksessa *2020 International Conference on Computer Engineering, Network, and Intelligent Multimedia (CENIM)*, 2020, s. 144–148. DOI: 10.1109/CENIM51130.2020.9297889.

[9] B. Ertl, Wikimedia Commons. (2015), url: https://commons.wikimedia.org/wiki/File:Euclidean_Voronoi_diagram.svg (viitattu 28. 01. 2025).

[10] R. J. Spick, P. Cowling ja J. A. Walker, ”Procedural Generation using Spatial GANs for Region-Specific Learning of Elevation Data”, teoksessa *2019 IEEE Conference on Games (CoG)*, 2019, s. 1–8. DOI: 10.1109/CIG.2019.8848120.

[11] H. Schott, E. Galin, E. Guérin, A. Paris ja A. Peytavie, ”Terrain Amplification using Multi Scale Erosion”, vol. 43, nro 4, 2024, ISSN: 0730-0301. DOI: 10.1145/3658200. url: https://doi.org/10.1145/3658200.