Ilari Paananen

Planetaarisen mittakaavan maaston generointi ja reaaliaikainen renderöinti

Tietotekniikan pro gradu -tutkielma

5. kesäkuuta 2019

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Ilari Paananen

Yhteystiedot: ilari.k.paananen@student.jyu.fi

Ohjaajat: Paavo Nieminen ja Tuomo Rossi

Työn nimi: Planetaarisen mittakaavan maaston generointi ja reaaliaikainen renderöinti

Title in English: Real-Time Planet Rendering

Työ: Pro gradu -tutkielma

Suuntautumisvaihtoehto: Ohjelmistotekniikka

Sivumäärä: 62+8

Tiivistelmä: Planeettojen renderöintiä hyödynnetään mm. viihdeteollisuudessa, avaruustutkimuksessa ja erilaisissa visualisoinneissa. Jotkut sovelluskohteet vaativat renderöinniltä reaaliaikaisuutta. Monesti planeettaa mallinnettaessa hyödynnetään olemassa olevia datajoukkoja, kuten satelliittien avulla Maasta saatuja korkeus- ja värikarttoja. Toisinaan kuitenkin halutaan renderöidä kuvitteellinen planeetta, josta ei entuudestaan ole olemassa dataa. Tällöin planetaarisen maaston renderöintiin tarvittavat datajoukot tulee luoda esimerkiksi proseduraalisesti. Koko planeetan kattava yksitoiskohtainen maasto vaatii valtavat määrät dataa, joten maastosta haluttaisiin generoida ja pitää tallessa vain sen verran kuin kulloinkin on tarpeen. Myös suuret etäisyydet aiheuttavat omat haasteensa planetaarisen mittakaavan renderöintiin. Tässä tutkielmassa taustoitetaan reaaliaikaista planeettarenderöintiä, esitellään joitain planetaarisen maaston generointiin soveltuvia menetelmiä, sekä kuvaillaan suuresta mittakaavasta aiheutuvia ongelmia ja niiden ratkaisuja. Lisäksi esitetään kolme, hieman toisistaan poikkeavaa planetaarisen maaston reaaliaikaiseen renderöintiin suunnattua menetelmää, jotka kehitettiin osana tätä tutkimusta.

Avainsanat: planeetta, renderöinti, reaaliaikainen, tietokone grafiikka, proseduraalinen, generointi

Abstract: Planet rendering is used in, for example, the entertainment industry, space research, and different kinds of visualisations. Some applications require that the rendering

happens in real-time. Often, to model a planet, preexisting datasets, such as height and color maps of Earth collected by satellites, are used. However, sometimes the planet to be rendered is a fictive one without any preexisting data. This is when the datasets needed to render the planetary terrain have to be created, for example, procedurally. Highly detailed terrain, that covers the whole planet, requires a huge amount of data. That's why it would be preferable to generate and store only as much terrain as is necessary at given time. Also, great distances innate to planetary scale rendering bring challenges of their own. In this thesis we give some background to realtime planet rendering, explain a few methods applicable to planetary terrain generation, and describe problems arising from the huge distances and show some solutions to those. In addition, we present three slightly differing methods designed for real-time planetary terrain rendering that were developed as part of this research.

Keywords: planet, rendering, real-time, computer graphics, procedural, generation

Termiluettelo

Indeksipuskuri	(engl. *index buffer*) on yleensä näytönohjaimella sijaitseva muistipuskuri, joka sisältää indeksejä kärkipistepuskuriin. Indeksit kertovat, miten kärkipisteistä muodostetaan monikulmioita piirtoa varten.
Kolmioverkko	(engl. *triangle mesh*) on kärkipisteistä ja niiden välille muodostuvista kolmioista koostuva kappale.
Kärkipiste	(engl. *vertex*) tarkoittaa tietokonegrafiikassa monikulmion kärkipistettä, jolla voi olla sijainnin lisäksi erilaisia ominaisuuksia, kuten väri ja normaalivektori.
Kärkipistepuskuri	(engl. *vertex buffer*) on yleensä näytönohjaimella sijaitseva muistipuskuri, joka sisältää kärkipisteitä.
Kärkipistevarjostin	(engl. *vertex shader*) on kärkipisteiden käsittelyyn tarkoitettu näytönohjaimella suoritettava ohjelma.
Laskentavarjostin	(engl. *compute shader*) on yleiskäyttöinen näytönohjaimella suoritettava ohjelma.
Pikselivarjostin	(engl. *pixel shader*) on pikseleiden väriarvojen laskemiseen tarkoitettu näytönohjaimella suoritettava ohjelma.
Planeetta	on tässä tutkielmassa suuri pallomainen taivaankappale, kuten planeetta, kääpiöplaneetta, kuu tai suuri asteroidi.
Rajaustilavuus	(engl. *bounding volume*) on jonkin alueen tai kappaleen sisäänsä sulkeva muoto, kuten rajauslaatikko (engl. *bounding box*) tai rajauspallo (engl. *bounding sphere*). Rajaustilavuuksia käytetään yleensä yksinkertaistamaan jotain toimenpidettä. Esimerkiksi kappaleiden näkyvyyden selvittäminen on usein helpompaa ja tehokkaampaa käyttäen kappaleiden rajaustilavuuksia kuin itse kappaleita.
Yksityiskohtaisuuden tasolla	(engl. *level of detail, LOD*) tarkoitetaan esimerkiksi kolmiulotteisen kappaleen piirron tehostamista vähentämällä yksityiskohtia yleensä etäisyyteen perustuen.

Matemaattiset merkinnät

$\mathbb{R}$	on reaalilukujen joukko.
$\mathbb{R}^n$	on n-ulotteisten reaalivektorien joukko.
s	on reaaliluku, eli $s \in \mathbb{R}$.
$\|s\|$	on reaaliluvun s itseisarvo.
f$(...)$	on reaalilukuarvoinen funktio, jonka parametrit annetaan sulkeiden sisällä pilkulla eroteltuina, esimerkiksi lerp(a,b,t).
$\boldsymbol{v}$	on reaalivektori, eli $\boldsymbol{v} \in \mathbb{R}^n$.
v_i	on vektorin $\boldsymbol{v}$ i:nnes komponentti.
$\boldsymbol{u} \cdot \boldsymbol{v}$	on vektorien $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$ välinen pistetulo: $\boldsymbol{u} \cdot \boldsymbol{v} = \sum_{i=1}^{n} u_i v_i$.
$\|\boldsymbol{v}\|$	on vektorin $\boldsymbol{v}$ normi eli pituus: $\|\boldsymbol{v}\| = \sqrt{\boldsymbol{v} \cdot \boldsymbol{v}}$.
$[a,b]$, $[a,b[$, $]a,b]$ ja $]a,b[$	ovat järjestyksessä suljettu väli, oikealta puoliavoin väli, vasemmalta puoliavoin väli ja avoin väli; a on välin alaraja ja b välin yläraja.
(s,t)	on kahden reaaliluvun muodostama monikko (engl. *tuple*).
$\mathbb{Z}$	on kokonaislukujen joukko.
$\mathbb{Z}^n$	on n-ulotteisten kokonaislukuvektorien joukko, eli kaikki ne vektorit $\boldsymbol{z}$, joiden jokainen komponentti $z_i \in \mathbb{Z}$.

Kuviot

Sisältö

1 Johdanto

Virtuaalisten kolmiulotteisten maastojen käyttökohteet ovat monenlaiset. Maastoja hyödynnetään esimerkiksi kolmiulotteisissa animaatioissa, elokuvissa, tietokonepeleissä, erilaisissa simulaatioissa sekä visualisoinneissa. Joissain kolmiulotteisten maastojen sovelluksissa, kuten elokuvissa ja animaatioissa, painotetaan renderöinnin nopeuden sijaan visuaalista näyttävyyttä. Kuitenkin useat käyttökohteet, kuten lentosimulaatiot ja pelit, ovat reaaliaikaisia, joten niin on oltava myös maastototeutuksen.

Kolmiulotteisia maastoja on tutkittu kauan, ja niiden toteutukseen sekä visualisointiin on kehitetty paljon erilaisia menetelmiä. Monet menetelmät perustuvat korkeuskarttoihin (engl. *height map, height field*), jotka renderöidään kolmioverkkoina (engl. *triangle mesh*). Tällaisia menetelmiä ovat kehittäneet esimerkiksi Berg ja Dobrindt (1995), Lindstrom ym. (1996), Duchaineau ym. (1997), Losasso ja Hoppe (2004) sekä Brodersen (2005). Reaaliaikaisuuden saavuttamiseksi nämä menetelmät pyrkivät vähentämään piirrettävien kolmioiden määrää, kuitenkin siten, että renderöitävä kolmioverkko ei poikkeasi liikaa korkeuskartan esittämästä maastosta.

Kolmioverkkojen ohella maastojen visualisointiin on sovellettu myös säteenheittoa (engl. *ray casting*), kuten Musgrave, Kolb ja Mace (1989), Cohen-Or ym. (1996), Qu ym. (2003) sekä Mantler ja Jeschke (2006). Vaikka kolmioverkkoihin perustuvat algoritmit ovatkin edelleen suosittuja reaaliaikaisissa sovelluksissa, suorituskykyinen maastojen renderöinti voidaan toteuttaa myös säteenheitolla, kun hyödynnetään viime vuosina yhä tehokkaammiksi käyneitä näytönohjaimia. Dick, Krüger ja Westermann (2009) esittävätkin varsin tehokkaan näytönohjainta hyödyntävän säteenheittomenetelmän.

Useimmat maastoalgoritmit on alkujaan suunniteltu tasomaisille maastoille. Pallomaisten maastojen, erityisesti planeettojen, renderöintiin soveltuville algoritmeille on myös tarvetta. Planeettoja on renderöity jo pitkän aikaa esimerkiksi tieteisfiktioaiheisissa televisiosarjoissa, elokuvissa sekä tiededokumenteissa. Viime vuosina on tullut yhä enenevissä määrin tarvetta myös planeettojen reaaliaikaiselle renderöinnille. Reaaliaikaisia planeettoja käyttävät esi-

merkiksi erilaiset visualisointi ja simulointi ohjelmistot, kuten Celestia [1], NASA WorldWind [2] ja Google Earth [3]. Reaaliaikaisia planeettoja nähdään yhä enemmän myös tietokonepeleissä, kuten Elite Dangerous [4] ja Star Citizen [5]. Planeettojen renderöintiin käytettävät algoritmit on yleensä kehitetty alkujaan tasomaastoille suunnattujen algoritmien pohjalta. Esimerkiksi Cignoni ym. (2003), Clasen ja Hege (2006), Kooima (2008) ja Porwal (2013) ovat kehitelleet tällaisia planeettojen renderöintiin soveltuvia menetelmiä.

Sekä tasomaastojen että planeettojen renderöinnissä käytetyt korkeusarvot voivat olla peräisin oikeista maastojen geologisista mittauksista saaduista datajoukoista, digitaalisista korkeusmalleista (engl. *digital elevation model, DEM*). Jotkin tällaisista datajoukoista kattavat kokonaisen planeetan, kuten esimerkiksi maapallon kattava NASA:n Blue Marble Next Generation [6]. Koska maastojen korkeusdatajoukot saattavat olla hyvin suuria, jopa useita gigatavuja, niitä ei aina voida pitää kokonaisuudessaan muistissa. Tällöin vain kulloinkin tarvittava data on luettava esimerkiksi kiintolevyltä. Losasso ja Hoppe (2004) esittävät keinon suurtenkin korkeuskarttojen tehokkaalle pakkaamiselle ja maaston reaaliaikaisessa renderöinnissä tarvittavien osien purkamiselle ajon aikana.

Oikeiden digitaalisten korkeusmallien sijaan maaston korkeusdata voidaan luoda myös erilaisilla proseduraalisilla menetelmillä. Proseduraalisille generointimenetelmille on tarvetta erityisesti tietokonepeleissä, joissa pelimaailman maastolle ei ole olemassa valmista korkeusmallia, tai koko maastoa ei ole järkevää luoda käsin esimerkiksi käyttäen pelin kenttäeditoria. Miller (1986) kuvailee eräitä fraktaalien maastojen proseduraaliseen generointiin soveltuvia algoritmeja ja esittää lisäksi oman menetelmänsä. Musgrave, Kolb ja Mace (1989) hyödyntävät Perlin-kohinaa (Perlin 1985) ja eri kohinatasojen summaamista sekä vesi- ja lämpöeroosion mallinnusta luonnollisen maaston generoinnissa. Archer (2011) kuvailee useita erilaisia korkeuskarttojen generoinnissa hyödynnettäviä proseduraalisia algoritmeja.

1. Celestia. Lisätietoa: `https://celestia.space/`.
2. NASA WorldWind. Lisätietoa: `https://worldwind.arc.nasa.gov/`.
3. Google Earth. Lisätietoa: `https://www.google.com/earth/`.
4. Elite Dangerous. Lisätietoa: `https://www.elitedangerous.com/`.
5. Star Citizen. Lisätietoa: `https://robertsspaceindustries.com/star-citizen`.
6. NASA Blue Marble Next Generation. Lisätietoa: `https://earthobservatory.nasa.gov/features/BlueMarble`.

Maastojen korkeuskartat voidaan esigeneroida, jolloin niitä voidaan käyttää samalla tavoin kuin digitaalisista korkeusmalleista saatuja korkeuskarttoja. Toisinaan esigenerointi ei ole järkevää tai edes mahdollista. Esimerkiksi tietokonepeli saattaa sisältää kokonaisen planeetan tai mahdollisesti useita planeettoja, joita pelaaja voi tutkia vapaasti. Yksityiskohtaisten korkeuskarttojen esigenerointi tällaisessa pelissä olisi järjetöntä, sillä koko planeetan kattavat korkeuskartat voivat vaatia suunnattoman määrän tallennustilaa. Parempi ratkaisu olisi generoida korkeusdataa aina tarpeen vaatiessa ajon aikana.

Losasso ja Hoppe (2004) yhdistävät menetelmässään oikean maailman korkeusdataa ja proseduraalista generointia. He lisäävät ajon aikana maastoon hieman proseduraalista korkeusvaihtelua, jotta katselijan ollessa hyvin lähellä maastoa siinä näkyisi vielä mielenkiintoisia yksityiskohtia, vaikka pohjalla olevassa korkeuskartassa niitä ei riittäisi.

Korkeusdatan valtavasta määrästä ja monien sovellusten kaipaamasta reaaliaikaisuudesta aiheutuvat haasteet eivät ole ainoita planetaaristen kappaleiden renderöinnissä. Myös valtavat etäisyydet tuovat mukanaan omat ongelmansa. Esimerkiksi 32-bittisten liukulukujen tarkkuus ei yksinkertaisesti riitä planetaarisella mittakaavalla. Samoin kolmiulotteisessa tietokonegrafiikassa yleensä käytetty syvyyspuskuri (engl. *depth buffer*) menettää nopeasti tarkkuutensa suurilla etäisyyksillä.

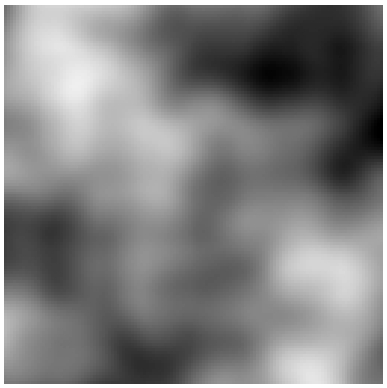
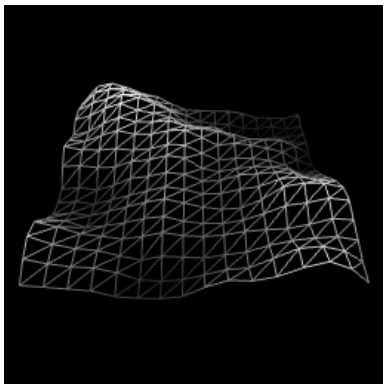
Tässä tutkielmassa keskitytään planetaarisen kappaleen korkeuskarttoja hyödyntävän maaston proseduraaliseen generointiin ja reaaliaikaiseen renderöintiin. Luvussa kaksi käsitellään kolmiulotteisia maastoja ja kuvaillaan joitain tasomaisten ja pallomaisten maastojen renderöintiin kehitettyjä menetelmiä. Korkeuskarttojen proseduraalista generointia käsitellään luvussa kolme ja suurten mittakaavojen aiheuttamia ongelmia sekä niiden ratkaisuja luvussa neljä. Luvussa viisi esitellään kolme hieman toisistaan poikkeavaa planeettojen renderöintiin suunnattua menetelmää, jotka kehitettiin osana tätä tutkimusta. Luvussa kuusi näitä kolmea menetelmää verrataan keskenään. Luku seitsemän on tutkielman yhteenveto.

2 Kolmiulotteiset maastot

Tässä luvussa tehdään nopea katsaus tasomaastoihin ja esitellään tarkemmin eräs alunperin tasomaastojen renderöintiin suunniteltu menetelmä. Tämän jälkeen kuvaillaan hieman pallomaisia maastoja ja erilaisia mahdollisia pallomaastojen lähtögeometrioita sekä kerrotaan joistain lähdekirjallisuudessa esitetyistä planeettojen renderöintimenetelmistä.

2.1 Tasomaastot

Kolmiulotteisten maastojen renderöintiä on tutkittu kauan. Perinteisesti maastojen renderöinnissä on hyödynnetty korkeuskarttoja (engl. *height map*). Yleensä korkeuskartat toteutetaan kaksiulotteisina taulukoina, joiden alkiot esittävät maaston pinnan korkeusvaihteluita. Renderöintiä varten maastolle muodostetaan kolmioverkko luomalla kärkipisteistä ja kolmioista ruudukko esimerkiksi *xy*-tasoon ja siirtämällä kärkipisteitä *z*-akselin suuntaisesti korkeuskartasta luetun korkeusarvon mukaisesti. Kuviossa 1 on esitetty korkeuskartta harmaasävykuvana ja sitä vastaava maasto kolmioverkkona.

a)  b) 

Kuvio 1. Korkeuskartta ja kolmioverkko. Kuvan (a) korkeuskartta on renderöity kolmioverkkona kuvassa (b).

Tällaiset korkeuskarttoihin perustuvat tasomaastot ovat rajoittuneita, koska maaston muodot perustuvat vain yhden akselin suuntaisiin poikkeamiin. Menetelmällä itsessään ei siis voida esittää luolia eikä ulkonemia. Tasomaastot ovat kuitenkin hyvin suosittuja, koska ne ovat yksinkertaisia toteuttaa ja niiden tehokkaaseen renderöintiin on kehitetty monia erilaisia me-

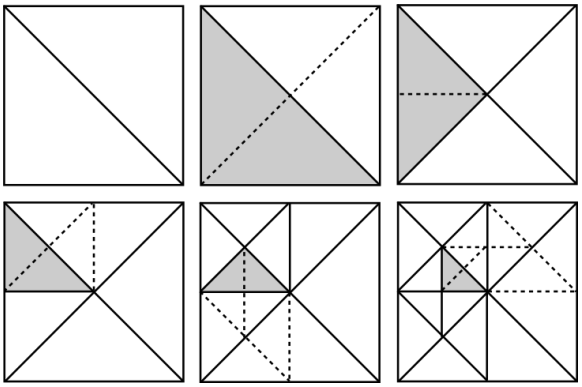
netelmiä. Ilman näitä menetelmiä piirrettäviä kolmioita tulisi hyvin paljon, sillä mielekkään maailman mallintamiseksi tarvitaan yleensä suuri ja yksityiskohtainen maasto.

Kolmioiden määrää voidaan vähentää säätelemällä maaston yksityiskohtaisuutta etäisyyteen perustuen. Kaukana katsojasta olevat yksityiskohdat ovat niin pieniä, ettei niitä voi kunnolla nähdä. Siispä kauempana maaston piirtoon voidaan käyttää vähemmän kolmioita ilman, että menetetään liikaa havaittavia yksityiskohtia. Tätä kutsutaan etäisyyteen perustuvaksi yksityiskohtaisuuden tasoksi (engl. *level of detail, LOD*).

2.2 ROAM-algoritmi

Eräs maaston renderöintiin suunniteltu LOD-algoritmi on *real-time optimally adapting meshes, ROAM*, jonka esittivät Duchaineau ym. (1997). Algoritmi perustuu binääripuuhun, jonka solmut ovat tasakylkisiä suorakulmaisia kolmioita. Kuvio 2 havainnollistaa ROAM-algoritmin toimintaa. Tavanomaisesti algoritmi aloitetaan kahdesta kolmiosta, joilla on yhteinen hypotenuusa. Nämä ovat puun ensimmäiset lehtisolmut, joille lasketaan virhearvo. Solmu, jolla on suurin virhe, jaetaan kahdeksi lapsisolmuksi kolmion hypotenuusan puolittajan kautta. Näille uusille lehtisolmuille lasketaan myös virhearvo, ja jako-operaatio suoritetaan jälleen kaikista lehtisolmuista sille, jolla se on suurin. Tätä jatketaan, kunnes virhe on riittävän pieni, lehtisolmuja on enimmäismäärä tai puun muodostamiseen varattu aika on käytetty. Lopullisen binääripuun lehtisolmujen kolmiot muodostavat maaston kolmioverkon, jonka kärkipisteiden korkeusarvot luetaan korkeuskartasta.

Jotta kolmioverkkoon ei tulisi rakoja, solmun jaon yhteydessä on jaettava myös yhteisen hypotenuusan omaava naapurisolmu. Mikäli solmun hypotenuusa onkin naapurisolmun toinen kateetti, pitää naapurisolmulle ensin suorittaa rekursiivisesti perus jako-operaatio. Yhden solmun jako voi siis vaatia usean muun solmun jaon. Koska maaston kolmioverkko ei yleensä muutu kovin paljon perättäisten kuvapiirtojen välillä, algoritmin tehostamiseksi Duchaineau ym. hyödyntävät jo luotua binääripuuta myöhemmillä piirtokerroilla jakamalla edelleen ja tarpeen tullen yhdistämällä sen lehtisolmuja.



Kuvio 2. ROAM-algoritmi vaiheittain. Lähtötilanteessa kolmioita on kaksi. Kulloinkin jaettavaksi valittu kolmio on korostettu harmaalla. Jaosta seuraava kolmiointi on esitetty katkoviivoin.

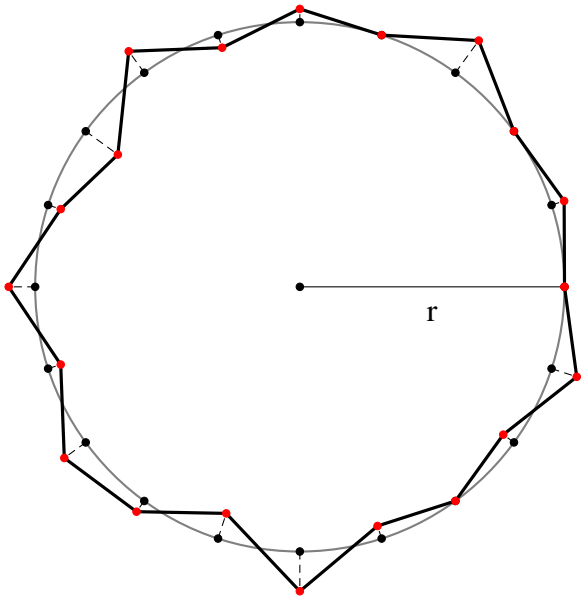
2.3 Pallomaiset maastot

Tasomaastot soveltuvat hyvin tilanteisiin, joissa virtuaalimaailma on niin pieni, ettei maastossa tarvitse huomioida planeetan pyöreyttä. Jos virtuaalimaailman halutaan kattavan huomattava osa planeetan pintaa, on tilanne toinen. Erityisesti, mikäli maaston tulisi esittää koko planeettaa, eivät tasomaastot toimi sellaisinaan. Tarvitaan maasto, jonka muoto on pohjimmiltaan pallomainen.

Kuviossa 3 on esitetty pallomaisen maaston periaate. Olkoon origo pallon keskipisteessä. Ensin maaston muodostavat kärkipisteet on vietävä pallon pinnalle. Tämä onnistuu kertomalla kärkipisteiden pallonormaaleja pallon säteellä r. Pallonormaalilla tarkoitetaan kärkipisteen suuntaista yksikkövektoria. Pallon sädettä kutsutaan myöhemmin myös planeetan säteeksi.

Pallon pinnalla olevat kärkipisteet on esitetty kuviossa mustina pisteinä. Seuraavaksi näitä siirretään pallonormaalien suuntaisesti korkeuskartasta luetun korkeusarvon verran. Korkeusarvot ovat siis poikkeamia planeetan säteestä. Nämä poikkeamat on esitetty katkoviivoin. Punaiset pisteet esittävät korkeusarvojen mukaan siirrettyjä kärkipisteitä. Maaston pinta muodostuu kärkipisteden välille piirrettävistä kolmioista, joita paksut, punaisia pisteitä yhdistävät mustat viivat kuvastavat. Kuinka kärkipisteet alunperin luodaan, riippuu käytetystä menetelmästä.

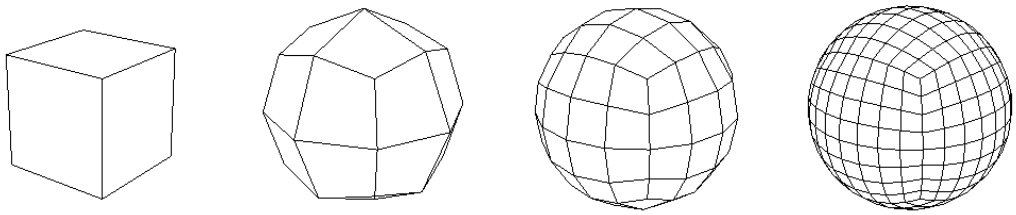
Yleensä pallomaisen maaston pohjalla on jokin monitahokas, jonka pisteet on viety pallon



Kuvio 3. Pallomaisen maaston periaate.

pinnalle. Tätä lähtögeometriaa aletaan sitten tihentää jakamalla tahkoja jollain tapaa pienemmiksi monikulmioiksi aina uudet pisteet pallon pinnalle vieden. Mitä useammin tihennys tehdään, sitä enemmän kappale muistuttaa palloa.

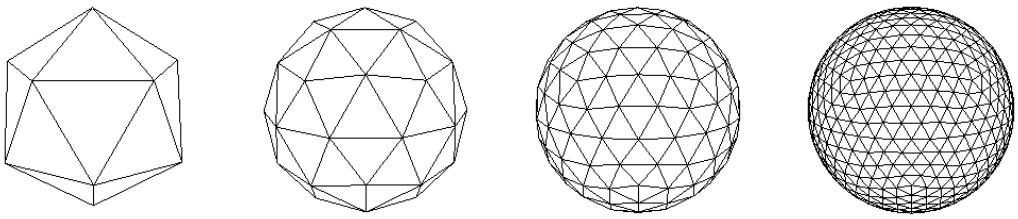
Kuution tihennys onnistuu helposti jakamalla nelikulmaiset tahkot aina neljäksi pienemmäksi nelikulmioksi. Kuvio 4 esittää näin tihennetyn kuution. Alkuperäisen kuution kulmapisteiden läheisyydessä nelikulmiot vääristyvät huomattavasti.



Kuvio 4. Kuutio ja sen kolme tihennystä.

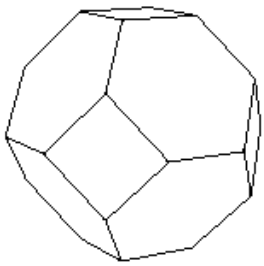
Säännöllinen ikosaedri (engl. *regular icosahedron*) on planeetan lähtögeometriaksi hyvin soveltuva monitahokas. Se muodostuu kahdestakymmenestä tasasivuisesta kolmiosta. Tämänkin kappaleen tihennys on helppoa: luodaan uudet pisteet kolmion sivujen puoleen väliin ja

muodostetaan neljä pienempää kolmiota näiden pisteiden ja alkuperäisen kolmion kulmapisteiden kautta. Tällä tavoin tihennetty ikosaedri on esitetty kuviossa 5. Tuloksena on hyvin tasainen kolmiointi, eikä kolmioissa näy juurikaan vääristymiä. Itse asiassa paras pallon pinnan kolmiointi seuraa juuri säännöllisen ikosaedrin tihennyksestä (Kooima 2008).



Kuvio 5. Säännöllisen ikosaedrin tihennys.

Eräs mielenkiintoinen monitahokas on Kelvinin rakenne (engl. *Kelvin structure*). Se koostuu kuudesta nelikulmiosta ja kahdeksasta kuusikulmiosta. Lähdekirjallisuudessa Kelvinin rakennetta ei ole mainittu käytetyn planeetan lähtögeometriana. Vaikkei se tuottaisikaan yhtä hyvää pallon pinnan kolmiointia kuin säännöllinen ikosaedri, olisi silti mielenkiintoista nähdä, miten se siihen soveltuu. Kelvinin rakenne on esitetty kuviossa 6.



Kuvio 6. Kelvinin rakenne.

Kooima (2008) renderöi virtuaalisen maapallon käyttäen planeettansa pohjalla säännöllistä ikosaedria. Ikosaedrin kolmioista hän muodostaa puuhierarkian, jota sitten tihennetään ROAM-algoritmiin perustuvalla menetelmällä. Kun näin muodostettu alustava kolmiointi on riittävän tiheä, Kooima renderöi jokaisen kolmion vielä tiheämpänä kolmioverkkona. Tämän kolmioverkon kärkipisteet lasketaan näytönohjaimella laskentavarjostimella. Jotta planeetan

pintaan ei jäisi rakoja eri kokoisten alustavien kolmioiden välille, Kooima valitsee kunkin kolmion renderöintiin käytettävän kolmioverkon siten, että se yhdistyy saumattomasti viereisiin kolmioihin.

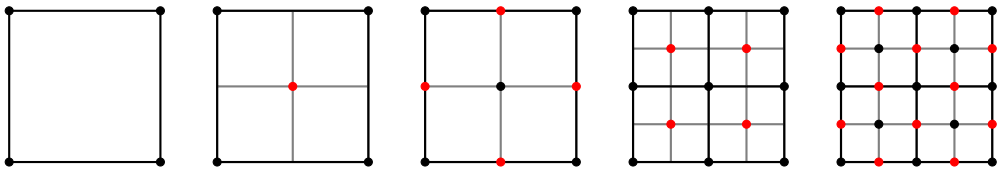
Porwal (2013) jakaa maapallon kuudeksi nelipuuksi. Hän tihentää nelipuita kameran sijainnin perusteella. Planeetan pinnan kärkipisteet muodostuvat nelipuun lehtisolmujen keski- ja kulmapisteisiin. Kolmiot piirretään näiden välille.

Clasen ja Hege (2006) soveltavat geometria leikekartta -menetelmää (*geometry clipmaps*, Losasso ja Hoppe 2004) planeetan renderöintiin. Siinä missä tasomaastoihin suunnitellut geometria leikekartat muodostavat sisäkkäisiä neliön muotoisia eri LOD-tason maastoalueita kameran ympärille, Clasen ja Hege levittävät planeetan pinnalle sisäkkäisiä renkaan mallisia eri LOD-tason vyöhykkeitä. He kutsuvat menetelmäänsä pallomaisiksi leikekartoiksi (engl. *spherical clipmaps*).

3 Korkeuskarttojen proseduraalinen generointi

Maaston korkeuskartan generoinnissa voidaan käyttää monenlaisia menetelmiä. Jotkin näistä menetelmistä vaativat, että mielivaltaisen pisteen korkeusarvoa määritettäessä pitää ensin laskea korkeusarvot ainakin osalle sen läheisyydessä olevista pisteistä. Toisinaan kaikki maaston korkeusarvot on määritettävä kerralla. Eräs tällainen menetelmä on rekursiiviseen jakoon perustuva timantti-neliö-algoritmi (engl. *diamond-square algorithm*) (Miller 1986).

Timantti-neliön toimintaperiaate on esitetty kuviossa 7. Ensin neliön kulmapisteet alustetaan esimerkiksi satunnaisilla arvoilla. Neliön keskipisteen arvo lasketaan kulmapisteiden keskiarvona, jota siirretään satunnaisella arvolla. Tätä kutsutaan neliö-vaiheeksi. Timantti-vaiheessa neliön jokaisen sivun puolittajalle lasketaan arvo sivun yhdistämien pisteiden ja sivun viereisten neliöiden keskipisteiden keskiarvona, johon jälleen lisätään satunnaissiirto. Jos sivun puolittaja on alkuperäisen neliön laidalla, keskiarvo voidaan laskea esimerkiksi vain kolmesta pisteestä. Näin saatiin kulmapisteet neljälle pienemmälle neliölle, joille sama voidaan toistaa neliö-vaiheesta lähtien. Satunnaissiirtojen vaikutusta pienennetään joka kierroksella.



Kuvio 7. Timantti-neliön toimintaperiaate. Ensimmäisessä kuvassa on neliö, josta algoritmi lähtee liikkeelle. Seuraavaat kaksi kuvaa ovat järjestyksessä ensimmäisen kierroksen neliö- ja timantti-vaiheet. Kaksi viimeistä kuvaa ovat toisen kierroksen vastaavat vaiheet. Punaiset pisteet esittävät kulloisessakin vaiheessa laskettavia pisteitä.

Jos maasto on hyvin suuri ja vaatii riittävän yksityiskohtaisuuden takaamiseksi paljon korkeusdataa, koko maaston kattavaa korkeuskarttaa ei ole järkevää generoida kerralla muun muassa tietokoneiden rajallisen muistikapasiteetin vuoksi. Tilanne on oletettavasti tämä esimerkiksi planetaarisen mittakaavan maastossa.

Tehdään suuntaa-antava laskelma suuren maaston korkeuskartan vaatimasta tallennustilasta.

Yksinkertaistuksen vuoksi oletetaan, että maasto on neliön muotoinen tasomaasto, jonka sivu on tuhat kilometriä [1]. Oletetaan myös, että maaston ruudukon välistys on yksi metri, ja että korkeuskartassa on jokaiselle ruudukon pisteelle oma korkeusarvonsa. Oletetaan vielä, että jokaisen korkeusarvon esittämiseen tarvitaan neljä tavua [2]. Näin ollen koko korkeuskartta vaatii tilaa noin $(10^6)^2 \cdot 4$ tavua $\approx 3,6$ teratavua. Planetaariset maastot voivat vaatia moninkertaisen määrän, useita kymmeniä teratavuja *per planeetta*.

Suuren maaston sovelluksissa korkeuskartasta halutaan generoida ajon aikana vain pieni, sopivan yksityiskohtainen osa kerrallaan. Miksi säilöä kaiken aikaa valtavaa ja yksityiskohtaista datajoukkoa, jos suuri osa maastosta ei näy kuvaruudulla lainkaan tai kattaa kuvaruudusta niin pienen alueen, etteivät yksityiskohdat ole havaittavia? Tarvitaan menetelmä, jolla korkeuskartasta voidaan tarvittaessa luoda vain haluttu osa halutulla tarkkuudella. Tähän soveltuvat kohinat (engl. *noise*).

Erilaiset kohina-algoritmit ovat olennaisia proseduraalisen sisällön luonnissa. Kohinoita käytetään esimerkiksi teksturoinnissa (mm. Perlin 1985), animoinnissa sekä erilaisten kappaleiden generoinnissa. Tässä tutkielmassa termillä *kohinafunktio* tarkoitetaan menetelmää, joka tuottaa mielivaltaiselle n-ulotteisen avaruuden pisteelle arvon ilman, että merkittävän monen muun pisteen arvoa tarvitsee laskea. Tämän määritelmän mukaan esimerkiksi aiemmin mainittu timantti-neliö-algoritmi ei ole kohinafunktio. Kohinafunktio antaa siis syötteenä saamalleen pisteelle $\boldsymbol{p}$ kohina-arvon v:

$$v = \text{noise}(\boldsymbol{p})$$

Ehkä yksinkertaisin kohinafunktio on arvokohina (engl. *value noise*). Avaruuteen määritetään ruudukko, jonka pisteille annetaan satunnaiset arvot. Kohina-arvo näissä pisteissä on kyseinen satunnaisarvo. Muissa pisteissä kohina-arvo saadaan interpoloimalla pistettä lähinnä olevien ruudukon pisteiden satunnaisarvoja. (Archer 2011)

Mandelbrot (1982) huomasi, että luonnossa esiintyvät muodot, kuten vuoristojen huiput, noudattelevat tiettyä kaavaa. Hän esitti, että oikealla maastolla on fraktaaleja piirteitä. Tä-

1. Planetaariset maastot voivat olla hyvinkin paljon suurempia, esimerkiksi maapallon ympärysmitta on noin neljäkymmäntätuhatta kilometriä.

2. Yhden 32-bittisen liukuluvun verran.

mä tarkoittaa sitä, että maaston suuren mittakaavan muodot näyttävät toistuvan samankaltaisina yhä uudelleen aina pienemmässä mittakaavassa. Aiemmin mainitun timantti-neliöalgoritmin tuottama korkeuskartta on suoraan fraktaali (Miller 1986). Useilla kohinoilla ei kuitenkaan ole luonnostaan tätä ominaisuutta.

Seuraavaksi kuvaillaan eräs tunnetuimmista ja eniten käytetyistä kohinafunktioista, Perlin-kohina, sekä esitetään, kuinka korkeuskarttaan saadaan fraktaaleja ominaisuuksia summaamalla useita eri taajuuksisia kohinatasoja yhteen. Lisäksi kerrotaan eroosion mallintamisesta ja uskottavampien maastojen generoinnista.

3.1 Perlin-kohina

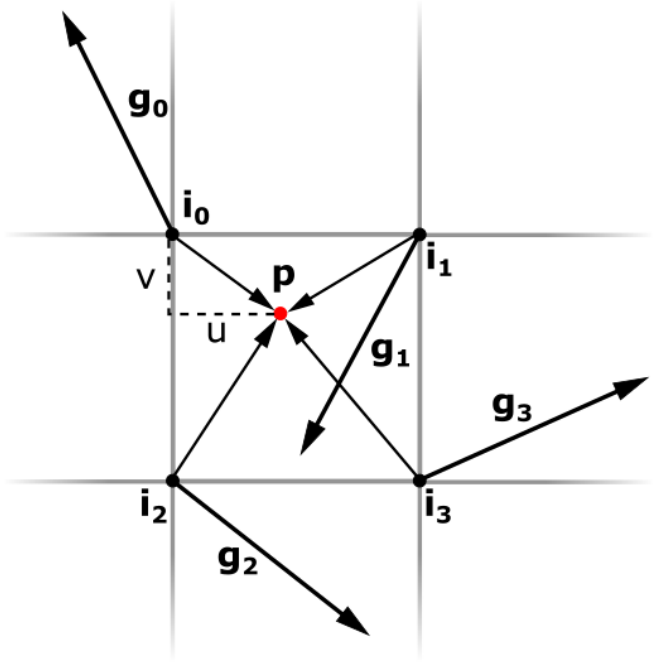
Perlin (1985) esitti eräänlaisen gradienttikohinan, eli kohinafunktion, jonka tuottamat kohina-arvot lasketaan avaruuteen määritetyn ruudukon pisteille annettujen pseudosatunnaisten gradienttivektorien avulla. Tämä sittemmin Perlin-kohinana tunnettu kohina käyttää ruudukkonaan kokonaislukuhilaa $\mathbb{Z}^n$ [3]. Kuvio 8 havainnollistaa, miten Perlin-kohina toimii kaksiulotteisessa avaruudessa. Sama periaate yleistyy kaikenulotteisiin reaalilukuavaruuksiin. Ensin etsitään pistettä $\boldsymbol{p}$ lähinnä olevat kokonaislukuhilan pisteet $\boldsymbol{i_0}, \boldsymbol{i_1}, \boldsymbol{i_2}$ ja $\boldsymbol{i_3}$ sekä niitä vastaavat gradienttivektorit $\boldsymbol{g_0}, \boldsymbol{g_1}, \boldsymbol{g_2}$ ja $\boldsymbol{g_3}$. Sitten lasketaan pistetulot $d_k = \boldsymbol{g_k} \cdot (\boldsymbol{p} - \boldsymbol{i_k})$, missä k saa arvot $0, 1, 2$ ja 3. Lopullinen kohina-arvo v saadaan sekoittamalla nämä pistetulot käyttäen sileää interpolointia (engl. *smooth interpolation*) seuraavasti:

$$
\begin{aligned}
d_{01} &= \mathrm{lerp}(d_0, d_1, \mathrm{s}(u)) \\
d_{23} &= \mathrm{lerp}(d_2, d_3, \mathrm{s}(u)) \\
v &= \mathrm{lerp}(d_{01}, d_{23}, \mathrm{s}(v))
\end{aligned}
$$

missä $\mathrm{lerp}(a, b, t) = (1-t)a + tb$ ja $\mathrm{s}(t) = 3t^2 - 2t^3$.

Perlin (2002) paranteli kohinaansa muun muassa käyttämällä edellä esitetyn kuutiollisen s-funktion sijasta funktiota $\mathrm{s}(t) = 6t^5 - 15t^4 + 10t^3$. Funktion ensimmäisen derivaatan lisäksi myös sen toisella derivaatalla on nollakohdat pisteissä $t = 0$ ja $t = 1$. Tämä ominaisuus vähentää esimerkiksi visuaalisia artefakteja eli virheitä tietyissä Perlin-kohinan sovelluksissa.

3. Kaikki ne n-ulotteisen avaruuden pisteet, joiden koordinaatit kuuluvat kokonaislukujen joukkoon $\mathbb{Z}$.



Kuvio 8. Perlin-kohinan periaate. Pisteen $\boldsymbol{p}$ kohina-arvo lasketaan kokonaislukusijainneissa $\boldsymbol{i_0}, \boldsymbol{i_1}, \boldsymbol{i_2}$ ja $\boldsymbol{i_3}$ määritettyjen pseudosatunnaisten gradienttivektorien $\boldsymbol{g_0}, \boldsymbol{g_1}, \boldsymbol{g_2}$ ja $\boldsymbol{g_3}$ avulla.

3.2 Fraktaali maasto

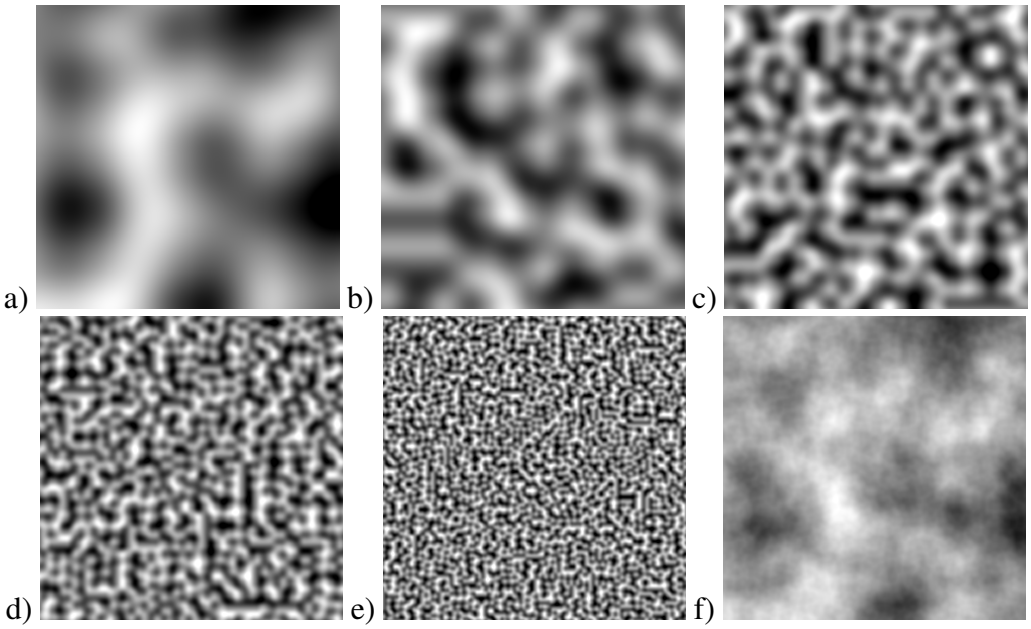
Luonnollisen maaston korkeuskartan luomiseksi kohinan tulisi toistua itsensä kaltaisena eri mittakaavoissa. Monen muun kohinafunktion tavoin Perlin-kohinallakaan ei ole tätä fraktaalia piirrettä. Se voidaan kuitenkin lisätä näytteistämällä kohinaa eri taajuuksilla ja summaamalla nämä kohinatasot eri painoarvoilla. Musgrave, Kolb ja Mace (1989) kutsuvat tällaista menetelmää kohinasynteesiksi (engl. *noise synthesis*). Perlin (1985) esitti vastaavan summakohinan, jossa kohinatason taajuus on kaksinkertainen edelliseen tasoon verrattuna ja painoarvo puolet. Summa voidaan esittää matemaattisesti muodossa

$$\sum_{i=1}^{n} \text{noise}(l^{i-1}\boldsymbol{p}) r^{i-1} \tag{1}$$

missä n on summattavien kohinatasojen määrä, l^{i-1} on kohinatason i taajuus ja r^{i-1} on kohinatason i painoarvo. Myös tämä summa on aiemman määritelmän mukainen kohinafunktio. Perlinin versiossa $l = 2$ ja $r = \frac{1}{2}$. Jotta yksittäisen kohinatason taajuus olisi edeltävän tason taajuutta korkeampi, tulee olla $l > 1$, ja puolestaan jotta tason painoarvo olisi edeltävän,

matalamman taajuuden kohinatason painoarvoa pienempi, tulee olla $0 < r < 1$.

Kuvio 9 havainnollistaa Perlin-kohinan summaamista vakioilla $n = 5$, $l = 2$ ja $r = \frac{1}{2}$. Mitä korkeampi näytteistystaajuus on, sitä tiheämmässä kohinan yksityiskohdat esiintyvät. Näiden eri taajuuksisten kohinatasojen voidaan ajatella olevan oikeassa maastossa havaittavia eri mittakaavan yksityiskohtia. Lopullinen kohina-arvo saadaan summaamalla kohinatasot siten, että matalan taajuuden tasoja painotetaan enemmän kuin korkean taajuuden.



Kuvio 9. Kohinatasojen summaaminen. Kuvissa (a)–(e) on viisi eri taajuuksista Perlin-kohinatasoa. Kuvassa (f) nämä kohinatasot on summattu eri painoarvoilla. Näin luodulla korkeuskartalla on fraktaaleja piirteitä.

3.3 Eroosio ja realistisemmat maastot

Vaikka edellä esitetyllä kohinatasoja summaavalla menetelmällä luotu maasto onkin luonteeltaan fraktaali, se ei kuitenkaan ole kovin realistinen. Tämä johtuu siitä, että menetelmä ei huomioi eroosion vaikutusta (Musgrave, Kolb ja Mace 1989). Oikea maasto on eroosion kuluttama. Eroosion irrottama maa-aines kulkeutuu painovoiman vaikutuksesta alas päin, vuoristoista laaksoihin, täyttäen koloja ja silottaen maaston epätasaisuuksia siellä, minne se asettuu. Korkealla vuoristoissa maasto onkin usein rosoisempaa ja korkeusvaihtelut ovat voi-

makkaampia kuin alempana.

Musgrave, Kolb ja Mace (1989) esittävät, kuinka kohinatasoja summattaessa voidaan approksimoida eroosion vaikutusta. He kuvailevat myös menetelmät vesi- ja lämpöeroosion mallintamiseen, mutta ne edellyttävät, että koko maaston korkeuskartta on luotu ennen menetelmien soveltamista. Tästä syystä näitä menetelmiä ei voida käyttää kaikissa sovelluksissa.

Pelkästään Perlin-kohinaa summaamalla saatu maasto on muodoiltaan "pehmeä". Se voisi sopia kumpuilevan maaston, kenties vuoriston aluskukkuloiden mallintamiseen, mutta itse vuoristoja se ei kuvasta kovin hyvin. Siinä ei ole vuoristoissa nähtäviä huippuja ja harjanteita. Yksinkertainen tapa luoda harjanteista kohinaa (engl. *ridged noise*) on ottaa kohinafunktion itseisarvo ja kertoa se miinus yhdellä. Tarpeen mukaan tätä arvoa voidaan vielä skaalata ja siirtää käyttämällä sopivia vakioita s ja t:

$$t - s|\,\mathrm{noise}(\boldsymbol{p})|$$

Huomaa, että tässä kohinafunktio noise($\boldsymbol{p}$) on kaavan 1 mukainen summakohina. Kuviossa 10 (a) on tällä tavoin Perlin-kohinalla luotu harjanteinen korkeuskartta. Itseisarvon jälkeen kohinafunktio saa pienimmän arvonsa nollakohdissaan, joissa funktio on myös epäjatkuva. Näissä kohdissa kohinafunktio vaihtaa yllättäen suuntaansa. Kun funktio käännetään kertomalla se miinus yhdellä, nollakohdista tulee funktion huippukohtia, joihin harjanteet muodostuvat.

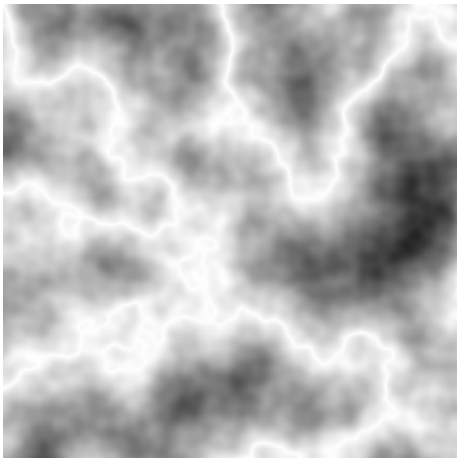
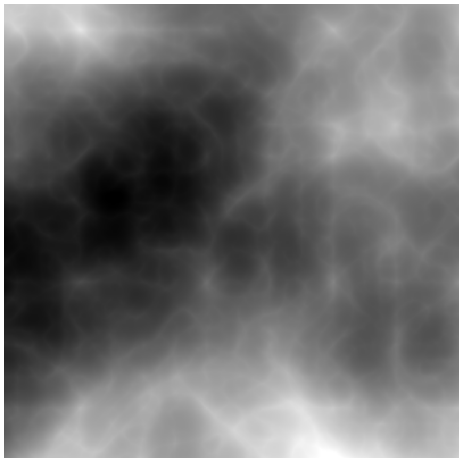
Seuraava kaava esittää toisenlaisen tavan luoda harjanteista kohinaa [4]. Perlin-kohinaa käyttäen tämä menetelmä tuottaa kuviossa 10 (b) esitetynlaisia korkeuskarttoja.

$$\begin{aligned} \mathrm{f}(\boldsymbol{p},0) &= 1 \\ \mathrm{f}(\boldsymbol{p},i) &= (1 - |\,\mathrm{noise}(l^{i-1}\boldsymbol{p})|)^2 \\ \sum_{i=1}^{n} \mathrm{f}(\boldsymbol{p},i)\,\mathrm{f}(\boldsymbol{p},i-1)r^{i-1} & \end{aligned} \tag{2}$$

Ensin esitetyssä harjanteisessa kohinassa itseisarvo ja kääntäminen tehtiin vasta kohinatasojen summaamisen jälkeen. Tässä puolestaan itseisarvo ja kääntö suoritetaan jokaiselle kohi-

4. Tämä tapa luoda harjanteista Perlin-kohinaa löytyy Sean T. Barretin ylläpitämästä avoimen lähdekoodin kirjastosta *stb_perlin.h*. Lisätietoa: `https://github.com/nothings/stb`.

natasolle erikseen. Kohinataso siirretään niin, että sen huippukohdat saavat arvon 1. Seuraavaksi arvo neliöidään, mikä korostaa huippukohtia ja tasoittaa arvoja nollan läheisyydessä. Tämä muistuttaa hieman eroosion vaikutusta. Lopuksi summattaessa kohinatasoa painotetaan erikoisesti muista esitetyistä summakohinoista tutun painoarvon lisäksi myös edellisen tason painottamattomalla kohina-arvolla.

a)  b) 

Kuvio 10. Kaksi erilaista harjanteista Perlin-kohinaa.

3.4 Yhdistelmäkohina

Kaavan 2 mukainen harjanteinen Perlin-kohina sopii vuoristomaastojen luontiin. Kutsutaan sitä vuoristokohinaksi. Planetaarisen maaston toteutuksen yhteydessä kuitenkin havaittiin, että jos koko maasto on luotu pelkästään tällä kohinalla, siitä ei tule kovin realistinen. Kun kaikkialla on samankaltaista vuoristoa, maasto käy tylsäksi. Vuorien lisäksi tarvitaan aluskukkuloita ja tasaisempaa maastoa.

Kaavan 1 mukainen Perlin-kohina soveltuu hyvin aluskukkuloiksi ja tasaisemmaksi maastoksi. Olkoo tähän tarkoitettu kohina nimeltään mäkikohina. Voisiko tätä ja vuoristokohinaa yhdistää siten, että joissain paikoin maasto on vuoristoa ja toisaalla mäkeä? Se onnistuu käyttämällä toista kaavan 1 mukaista kohinaa. Sanotaan sitä maskikohinaksi. Valitaan jokin kohina-arvojen väli $[a,b]$. Kun pisteelle määritetään yhdistelmäkohinan arvoa, ensin määritetään maskikohinalla arvo m kyseisessä pisteessä. Jos $m < a$, yhdistelmäkohinan arvo on mäkikohinan arvo h kyseisessä pisteessä. Jos $m > b$, yhdistelmäkohinan arvo on puo-

lestaan vuoristokohinan arvo k kyseisessä pisteessä. Jos $m \in [a,b]$, yhdistelmäkohinan arvo v saadaan sekoittamalla mäki- ja vuoristokohinoiden arvoja: $v = \text{lerp}(h,k,t)$, missä lerp on aiemmin esitetty interpolointifunktio ja $t = \frac{m-a}{b-a}$.

4 Suurten maailmojen ongelmat

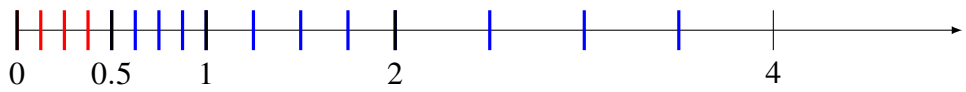
Nykyään tietokoneiden liukuluvut ovat lähes poikkeuksetta IEEE:n (Institute of Electrical and Electronics Engineers) liukulukuaritmetiikan standardin IEEE 754 mukaisia. Jotkin laskimet mahdollisesti käyttävät standardia IEEE 854, koska se sallii laskimille hyödyllisen kymmenkantaisen lukujärjestelmän käytön (Goldberg 1991). IEEE 754 standardi määrittää muun muassa 32-bittiset ja 64-bittiset liukuluvut. Tästä eteenpäin 32- ja 64-bittisillä liukuluvuilla viitataan IEEE 754 standardin mukaisiin liukulukuihin.

4.1 IEEE 754 standardin liukuluvut

Tämän tutkielman kannalta on oleellista ymmärtää jotain liukulukujen luonteesta. Seuraava selostus pohjautuu Goldbergin (1991) liukulukuaritmetiikkaa käsittelevään julkaisuun. Liukuluku koostuu neljästä osasta, jotka ovat etumerkki s, mantissa eli luvun merkitsevät numerot m, kantaluku B ja eksponentti e. Esimerkiksi luku -1230 voidaan esittää liukulukuna muodossa $-1.23 \cdot 10^3$. Tässä $s = -1$, $m = 1.23$, $B = 10$ ja $e = 3$. IEEE 754 standardin liukuluvut ovat kaksikantaisia, eli $B = 2$. Jotta jokaiselle liukuluvulle olisi vain yksi esitys, standardi määrittää, että lähes kaikki liukuluvut ovat normalisoituja (engl. *normalized numbers*). Tämä tarkoittaa sitä, että liukuluvun eniten merkitsevä numero on yksi. Kohta huomataan, että tämän ansiosta mantissan esittämisessä säästetään yksi bitti. Normalisoitujen liukulukujen lisäksi tarvitaan esitys nollalle. Nollia on itse asiassa kaksi: negatiivinen ja positiivinen nolla. Nollan ja sitä lähinnä olevan normalisoidun liukuluvun väliin jää aukko, joka täytetään niin kutsutuilla ei-normalisoiduilla [1] luvuilla (engl. *denormalized numbers*). Myös negatiiviselle ja positiiviselle äärettömyydelle (engl. *negative and positive infinity*) on omat esityksensä. Vielä on joukko epälukuja [2] (engl. *Not a Number, NaN*), jotka voivat seurata erilaisista virheellisistä laskutoimituksista. Lukuun ottamatta positiivista ääretöntä, kuviossa 11 on esitetty positiiviset liukulukuarvot kuvitteelliselle liukulukutyypille.

1. Käännös Wikipedian suomenkielisestä liukulukuja käsittelevästä artikkelista: `https://fi.wikipedia.org/wiki/Liukuluku` (luettu 23.3.2018).

2. Käännös Pekka Hotokan paperista "Liukulukulaskenta": `http://users.jyu.fi/~pejuhoto/opiskelu/liukuluvut.pdf`.



Kuvio 11. Positiiviset liukuluvut lukujanalla. Liukulukujen kantaluku $B = 2$, eksponentti $e \in \{-1, 0, 1\}$ ja mantissan merkitsevien numeroiden määrä on 3. Siniset viivat kuvaavat normalisoituja liukulukuja, punaiset ei-normalisoituja ja paksut numeroidut viivat nollaa sekä liukulukuja, joiden mantissa on 1.00. Tällä liukulukutyypillä ei voida esittää lukua neljä, mutta se on mukana havainnollistuksen vuoksi.

Reaalilukuja on äärettömän monta, joten kaikkia niitä ei voida esittää rajallisella määrällä bittejä. Etumerkin esittämiseen tarvitaan yksi bitti. Loput liukuluvun biteistä jaetaan mantissan ja eksponentin kesken. Kahdella eksponentin bittiesityksellä on erikoismerkitys. Kun kaikki eksponentin bitit ovat nollia, liukuluku esittää joko nollaa (kun $m = 0$) tai ei-normalisoitua lukua (kun $m \neq 0$). Kun kaikki eksponentin bitit ovat puolestaan ykkösiä, liukuluku esittää joko ääretöntä (kun $m = 0$) tai epälukua (kun $m \neq 0$). Kaikki muut eksponentit ovat normalisoiduille liukuluvuille. Koska normalisoidut ja ei-normalisoidut luvut voidaan erottaa toisistaan eksponentin perusteella, ja koska normalisoitujen lukujen eniten merkitsevä numero on aina yksi ja ei-normalisoitujen nolla, liukuluvun eniten merkitsevä numero voidaan jättää pois mantissan bittiesityksestä.

IEEE 754 standardin 32-bittisten liukulukujen mantissalle on varattu 23 bittiä ja eksponentille loput 8. Eksponentti on kokonaisluku väliltä $[-126, 127]$. Puolestaan 64-bittisissä liukuluvuissa mantissa on 52 bittiä ja eksponentti 11. Eksponentti on kokonaisluku väliltä $[-1022, 1023]$. On tärkeää havaita, että jokaista eksponenttia kohden on saman verran eri liukulukuja. Esimerkiksi liukulukuja, joiden eksponentti $e = 0$, on yhtä monta kuin liukulukuja, joiden eksponentti $e = 10$. Toisin sanoen välillä $[1, 2[$ on yhtä monta liukulukua kuin välillä $[1024, 2048[$. Liukulukuja on siis tiheämmin lähellä nollaa. Itseasiassa välillä $[0, 1[$ on *lähes yhtä monta* liukulukua kuin välillä $[1, \infty[$.

4.2 Sijaintien esittäminen

Nykyaikaiset näytönohjaimet kykenevät suorittamaan laskutoimituksia sekä 32- että 64-bittisillä liukuluvuilla. Kuitenkin näytönohjaimet laskevat 32-bittsillä liukuluvuilla

huomattavasti tehokkaammin kuin 64-bittisillä liukuluvuilla. Esimerkiksi NVIDIAn Pascal-arkkitehtuurin näytönohjain Tesla P100 kykenee 10.6 teraliukulukuoperaatioon sekunnissa (engl. *tera floating point operations per second, TFLOPS*) 32-bittisillä liukuluvuilla, mutta vain puoleen tästä 64-bittisillä liukuluvuilla (NVIDIA 2017). Tämän vuoksi, erityisesti jos laskettavaa on paljon, näytönohjaimella suoritettavien varjostinohjelmien on järkevää käyttää laskuissaan 32-bittisiä liukulukuja 64-bittisten sijaan. Lisäksi 64-bittiset liukuluvut vaativat tuplasti tallennustilaa 32-bittisiin nähden.

Mallinnettaessa planeettaa on luonnollista ajatella koordinaatiston origon olevan planeetan keskipisteessä. Planeetan piirtämiseksi luodaan kolmioverkko, jonka kärkipisteet ovat planeetan pinnalla. Yksityiskohtaisen planeetan esittämiseksi kolmioita tarvitaan paljon. Jotta näytönohjain voisi piirtää suuren määrän kolmioita tehokkaasti, halutaan kärkipisteiden koordinaatit esittää 32-bittisillä liukuluvuilla. Tästä kuitenkin seuraa ongelma: 32-bittisistä liukuluvuista loppuu tarkkuus planetaarisen mittakaavan etäisyyksillä.

Otetaan esimerkiksi maapallo. Sen keskisäde r on noin 6371 kilometriä. Olkoon koordinaatiston yksikkö metri ja origo planeetan keskipisteessä. Tällöin 32-bittisten liukulukukoordinaattien tulee esittää etäisyyksiä, jotka vastaavat suuruusluokaltaan maapallon sädettä. Nämä etäisyydet osuvat välille $[2^{22}, 2^{23}]$, sillä

$$2^{22} = 4194.304 \cdot 10^3 < r < 8388.608 \cdot 10^3 = 2^{23}$$

Tällä välillä on tasavälein $2^{23} + 1$ liukulukua, joista kaksi on välin päätepisteissä. Näin ollen vierekkäisten liukulukujen esittämien koordinaattien etäisyys on

$$\frac{2^{23} - 2^{22}}{2^{23}} = 0.5$$

metriä. Jos planeetta on yksityiskohtainen ja sitä tarkastellaan läheltä, puolen metrin kynnykset maaston korkeusvaihteluissa ovat luultavasti melko häiritseviä. Lisäksi kameran sijainnin esittäminen puolen metrin tarkkuudella ei liene kovin sulavaa.

Kooima (2008) ratkaisee ongelman generoimalla maaston kolmioverkon siten, että kameran sijainti toimii kärkipisteiden origona. Tämä voidaan tehdä esimerkiksi käyttämällä 64-bittisiä liukulukuja [3] kameran sijainnin ja maaston alustavan kolmioinnin kärkipisteiden sijaintien

3. 64-bittiset liukuluvut kykenevät esittämään maapallon säteen suuruusluokan lukuja alle nanometrin tarkkuudella, minkä luulisi riittävän vallan mainiosti.

esittämiseen origon ollessa planeetan keskipisteessä. Ennen varsinaisen kolmioverkon generointia kärkipisteet siirretään ensin kameran avaruuteen vähentämällä niistä kameran sijainti ja sitten kärkipisteiden koordinaatit muunnetaan 32-bittisiksi liukuluvuiksi. Nyt lähellä kameraa olevien kärkipisteiden 32-bittisillä liukuluvuilla esitetyt sijainnit ovat tarkkoja, koska liukulukujen tarkkuus sijoittuu nollan läheisyyteen. Toisaalta, mitä kauempana piste on kamerasta, sitä epätarkempi sen sijainti on. Tämä ei kuitenkaan haittaa, sillä etäisyyden kasvaessa virhe kuvaruudulla pienenee perspektiiviprojektiosta johtuen, eikä kaukana olevien kärkipisteiden sijaintien epätarkkuus ole havaittavissa.

4.3 Syvyyspuskuri ja Z-kilpailu

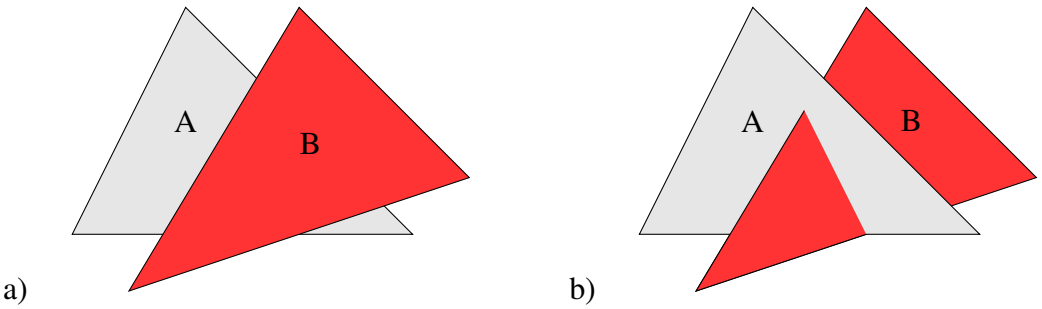
Määritetään ensin joitain apukäsitteitä. Näkymäpyramidi (engl. *view frustum*) on kolmiulotteisen näkymän rajat määrittävä pyramidin muotoinen alue. Sen kärki on kameran sijainnissa ja se kasvaa katselusuuntaan. Pyramidin huipun katkaisee näkymän etuseinä (engl. *near plane*). Yleensä myös pyramidi rajoittuu näkymän takaseinään (engl. *far plane*). Pisteen z-arvo näkymässä on pisteen z-koordinaatti kameran avaruudessa. Pisteen syvyysarvo puolestaan on verrannollinen pisteen etäisyyteen näkymän etuseinästä. Usein näkymän z-arvoa ja syvyysarvoa käytetään synonyymeinä. Hyvä. Sitten asiaan.

Monikulmioista koostuvia kolmiulotteisia kappaleita ei voida piirtää miten sattuu, jos lopullisen kuvan halutaan näyttävän oikeanlaiselta. Monikulmioita piirrettäessä on huomioitava kaksi seikkaa:

- Usein monikulmiot ovat kuvaruudulla päällekkäin. Tällöin lähempänä kameraa olevien monikulmioiden tulisi peittää kauempana olevia monikulmioita.
- Joskus monikulmiot voivat myös leikata toisiaan. Tällöin monikulmion halutaan piirtyvän osittain toisen eteen ja osittain taakse.

Kuviossa 12 on esitetty nämä kaksi tilannetta kahdella kolmiolla. Tämän kaltaisia tilanteita varten suunniteltuja menetelmiä näkyvien pintojen määrittämiseen (engl. *visible-surface determination*) on kehitetty paljon, joista useita muun muassa Foley ym. (1996) kuvailevat. Newell, Newell ja Sancha (1972) esittivät erään tällaisen algoritmin. Se takaa lopullisen kuvan oikeellisuuden molemmissa edellä esitetyissä tilanteissa. Menetelmässä monikulmiot

järjestetään syvyyden mukaan ja piirretään takaa eteen. Jos monikulmiot leikkaavat toisiaan, tai jos niitä ei muusta syystä voida laittaa syvyysjärjestykseen, pilkotaan ne ensin sellaisiin osiin, että järjestäminen onnistuu.



Kuvio 12. Kolmioiden piirtojärjestys. Kuvassa (a) kolmio A on kolmion B takana, ja kuvassa (b) kolmiot leikkaavat toisiaan.

Catmull 1974 kehitti tavan piirtää kolmiulotteisia kappaleita mielivaltaisessa järjestysessä menettämättä lopullisen kuvan oikeellisuutta. Menetelmä soveltuu myös monikulmioiden piirtoon. Pikselien väriarvot sisältävän niin kutsutun kuvapuskurin (engl. *frame buffer*) ohella pidetään syvyyspuskria (engl. *depth buffer*) [4], johon kunkin pikselin syvyysarvo tallennetaan. Aluksi syvyyspuskurin jokainen alkio alustetaan arvolla, joka vastaa näkymän suurinta mahdollista z-arvoa. Monikulmiota piirrettäessä jokaisen sen kattaman pikselin syvyysarvo lasketaan ja sitä verrataan syvyyspuskurissa olevaan arvoon. Mikäli syvyysvertailun mukaan monikulmion pikseli kuuluu piirtää puskureista löytyvän pikselin eteen, kuva- ja syvyyspuskureiden arvot ylikirjoitetaan monikulmion pikselin väri- ja syvyysarvoilla. Muutoin puskurit jätetään kyseiseisen pikselin osalta rauhaan, ja myös pikselin väriarvo voidaan jättää määrittämättä. Jos väriarvon määrittäminen on verrattain raskasta, monikulmioiden järjestäminen ja piirtäminen karkeasti syvyysjärjestyksessä edestä taakse voi olla hyödyllistä, koska tällöin peittoon jäävien pikselien väriarvoja ei todennäköisemmin tarvitse laskea turhaan (Foley ym. 1996).

Nykyään tietokonegrafiikassa hyödynnetään siihen tarkoitettua laitteistoa, näytönohjainta.

4. Syvyyspuskurista käytetään myös nimeä Z-puskuri (engl. *Z-buffer*).

Näytönohjainta käytetään jonkin grafiikkarajapinnan, kuten OpenGL:n [5] tai Direct3D:n [6], avulla. Näiden rajapintojen kautta saadaan käyttöön myös laitteistokiihdytetty syvyyspuskuri. Syvyyspuskuriin kirjoitettavat syvyysarvot ovat liukulukuja väliltä $[0,1]$ [7]. Tavanomaisesti syvyysarvo nolla vastaa näkymäpyramidin etuseinää ja yksi takaseinää. Perspektiiviprojektio aiheuttaa sen, että syvyysarvot ovat verrannollisia pisteen näkymäkoordinaatiston z-arvon käänteislukuun $\frac{1}{z}$ (Sellers ym. 2013). Jos z on lähellä nollaa, pienikin z-arvon muutos aiheuttaa suuren muutoksen sitä vastaavassa syvyysarvossa. Toisaalta jos z on kaukana nollasta, pieni muutos z-arvossa ei aiheuta juurikaan muutosta syvyysarvossa. Syvyysarvot ovat siis tarkempia lähellä nollaa. Ensin tämä voi vaikuttaa hyvältä, sillä tarkkuutta kaivataan kameran läheisyydessä. Käytännössä z-arvon kasvaessa syvyysarvojen tarkkuus heikkenee niin nopeasti, että kaksi suurestikin toisistaan poikkeavaa z-arvoa voivat saada saman syvyysarvon. Jos kaksi päällekkäin olevaa pistettä saavat saman syvyysarvon, se kumpi jää näkyviin riippuu piirtojärjestyksestä, ja toisinaan oikeasti taaempana oleva piste voi piirtyä etummaisen päälle. Tätä tilannetta kutsutaan Z-kilpailuksi (engl. *Z-fighting*). Kameran liikkuessa pisteiden z-arvot voivat yhdellä piirtokerralla pyöristyä eri syvyysarvoihin ja toisella samaan. Tällöin pikseli "vilkkuu", mikä on hyvin häiritsevää etenkin, kun yleensä tämä tapahtuu samaan aikaan useille pikseleille. Näkymäpyramidin takaseinän etäisyyden f suhde etuseinän etäisyyteen n vaikuttaa siihen, kuinka nopeasti syvyyspuskuri menettää tarkkuutensa (Sellers ym. 2013). Tilanne on erityisen paha planetaarisessa mittakaavassa, jossa suhdeluku f/n voi olla sadoissa miljoonissa tai jopa miljardeissa.

Ongelmaan on monia erilaisia ratkaisuja. Yksi niistä on näkymän jakaminen useampaan näkymäpyramidiin, joita käyttäen näkymä piirretään järjestyksessä takimmaisesta etummaiseen aina välissä syvyyspuskuri tyhjentäen. Jako pitää tehdä niin, että kaikkien pyramidien f/n-suhde on sopiva. Esimerkiksi kolmella näkymäpyramidilla voidaan kattaa näkymä yhdestä metristä miljoonaan kilometriin käyttäen (f,n)-pareja $(1,1000)$, $(1000,10^6)$ ja $(10^6,10^9)$. Tällöin jokaisen näkymäpyramidin f/n-suhde on 1000. (Sellers ym. 2013)

5. OpenGL. Lisätietoa: `https://www.opengl.org/`.

6. Direct3D on osa Microsoftin kehittämää DirectX-ohjelmistorajapintaa. Lisätietoa: `https://docs.microsoft.com/en-us/windows/desktop/direct3d`.

7. Syvyyspuskuri ei välttämättä sisällä suoraan liukulukuja, joten ennen varsinaista kirjoittamista syvyysarvot muunnetaan vielä syvyyspuskurille sopivaan muotoon.

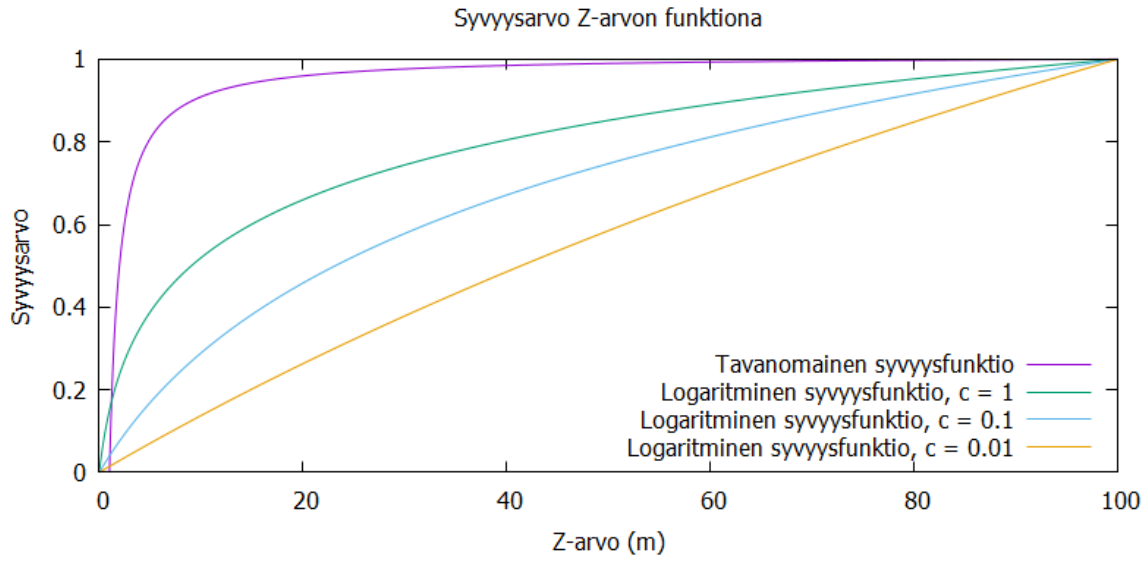
Toinen ratkaisu Z-kilpailuun on kirjoittaa syvyyspuskuriin logaritmisia syvyysarvoja. Kutsutaan syvyysfunktioksi sellaista funktiota g, joka antaa näkymän z-arvolle sitä vastaavan syvyysarvon. Tavanomainen syvyysfunktio, jolle $\mathrm{g}(n) = 0$ ja $\mathrm{g}(f) = 1$, on muotoa

$$\mathrm{g}(z) = \frac{(fz - fn)}{z(f - n)}$$

Kemen (2012) esittää logaritmisen syvyysfunktion, jolla syvyyspuskurin tarkkuus paranee niin, että Z-kilpailusta ei ole ongelmaa planetaarisella mittakaavallakaan:

$$\mathrm{g}(z) = \frac{\ln(zc+1)}{\ln(fc+1)}$$

missä ln on luonnollinen logaritmi. Säätämällä vakion c arvoa funktiosta saadaan tarpeen mukaan enemmän tai vähemmän lineaarinen. Kuviossa 13 on esitetty vakioilla $n = 1$ ja $f = 100$ tavanomainen syvyysfunktio sekä kolme logaritmista syvyysfunktiota eri vakion c arvoilla.



Kuvio 13. Erilaisia syvyysfunktioita.

Kuviosta huomataan, että tavanomainen syvyysfunktio käyttää suurimman osan syvyysarvoista näkymäpyramidin etuseinän lähellä oleviin z-arvoihin. Logaritmisilla syvyysfunktioilla syvyysarvot puolestaan jakautuvat paljon tasaisemmin, jolloin syvyyspuskurin tarkkuus riittää suurillakin etäisyyksillä. Kemenin (2012) logaritminen funktio tosin käyttää osan syvyysarvoista turhaan välin $[0, n[$ z-arvoille.

5 Toteutus

Osana tutkimusta kehitettiin kolme hieman toisistaan poikkeavaa planeettojen reaaliaikaiseen renderöintiin suunnattua menetelmää. Ensimmäisenä toteutettiin nelipuihin perustuva menetelmä, tai lyhyemmin, *nelipuumenetelmä*. Tämän pohjalta toteutettiin kolmioiden jakoon perustuva menetelmä. Viimeiseksi kehitettiin rajoitetumpaan kolmioiden jakoon perustuva menetelmä. Näistä kolmioihin perustuvista menetelmistä käytetään järjestyksessä nimityksiä *vapaa kolmiointi* ja *rajoitettu kolmiointi*.

Seuraavaksi kaikki kolme menetelmää kuvaillaan toteutusjärjestyksessä: ensin nelipuumenetelmä, sitten vapaa kolmiointi ja lopuksi rajoitettu kolmiointi. Koska menetelmät kehitettiin aina edellisen päälle, on luonnollista kuvailla kaikille menetelmille yhteiset osat nelipuumenetelmän yhteydessä. Samoin kahden kolmioihin perustuvan menetelmän jakamat ominaisuudet kuvaillaan vapaan kolmioinnin yhteydessä.

5.1 Nelipuumenetelmä

Ensimmäinen toteutettu planeettojen renderöintimenetelmä sai innoituksensa Kooiman (2008) ja Porwalin (2013) esittämistä menetelmistä. Se perustuu nelipuihin, kuten Porwalinkin (2013) menetelmä. Kooima (2008) luo planeetan pinnan alustavan kolmioinnin ROAM-algoritmin kaltaisella menetelmällä ja täyttää sitten nämä alustavat kolmiot piirrettäessä tiheämmillä kolmioverkoilla. Toteutetussa menetelmässä tällainen alustava geometria muodostuu kolmioiden sijaan nelipuiden nelikulmaisista lehtisolmuista.

5.1.1 Alustavan geometrian luonti

Planeetan geometrian luonti aloitetaan kuutiosta, jonka kulmapisteet ovat planeetan säteen etäisyydellä planeetan origosta. Kuution jokainen tahko toimii oman nelipuunsa juurisolmuna. Kuvio 14 havainnollistaa yhden nelipuun jako-operaatiota katselijan lähestyessä planeettaa. Jokainen nelipuu käydään läpi vuorollaan alkaen juurisolmusta. Mikäli solmu täyttää kohta esitettävän etäisyyteen perustuvan jakokriteerin, se jaetaan neljäksi lapsisolmuksi. Lapsisolmujen kulmapisteinä toimivat vanhempisolmun kulmapisteiden lisäksi vanhemman

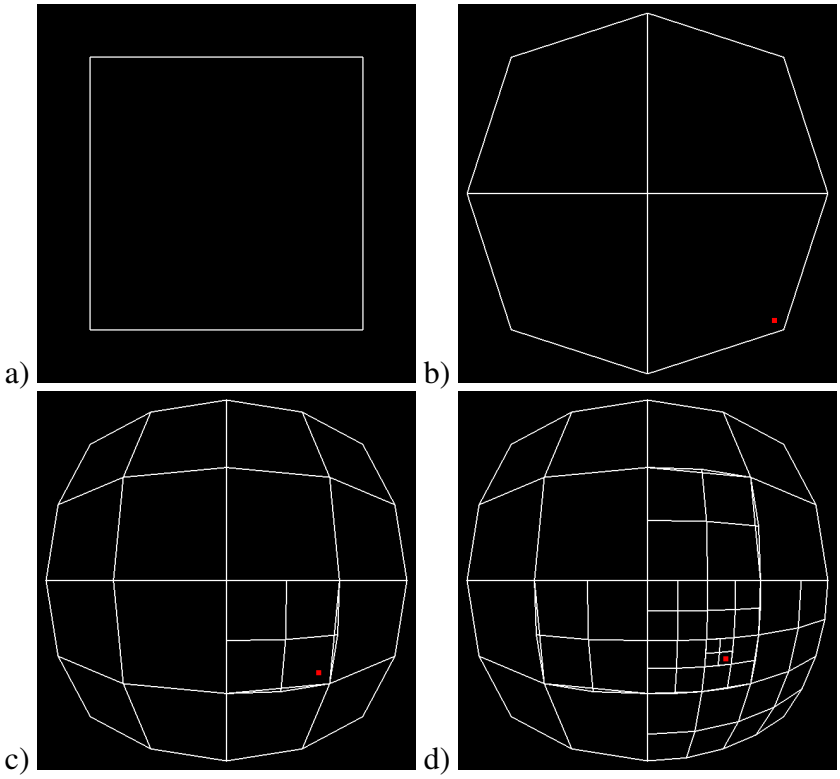
keskipiste ja vanhemman sivujen keskipisteet, jotka siirretään planeetan säteen etäisyydelle origosta. Sama toistetaan rekursiivisesti uusille lapsisolmuille, kunnes sopiva yksityiskohtaisuuden taso saavutetaan.

Sopivan yksityiskohtaisuuden tason määrittää jakokriteeri. Tässä toteutuksessa nelipuilla on maksimisyvyys, eli tämän syvyyden solmuja ei enää jaeta lapsisolmuiksi. Muutoin solmu jaetaan, jos katselijan etäisyys sille karkeasti määritetystä rajauslaatikosta (engl. *bounding box*) on vähemmän kuin rajauslaatikon kaksinkertainen lävistäjä. Rajauslaatikkoa määritettäessä tulisi huomioida solmun kattaman alueen korkeusarvot. Kaikkia näitä korkeusarvoja ei kuitenkaan ole välttämättä saatavilla eikä niitä haluta laskea vielä tässä vaiheessa. Rajauslaatikko luodaankin vain solmun keski- ja kulmapisteistä, joille korkeusarvot lasketaan.

Ulrich (2002) esittää tavan arvioida pintaa approksimoivan geometrian maksimivirhettä kuvaruudulla ja perustaa jakopäätöksen siihen. Menetelmä on suunniteltu perinteisten tasomaisten maastojen piirtoon ja vaatii kullekin nelipuun solmulle esilaskentaa. Solmulle lasketaan rajaustilavuus katselijan etäisyyden määrittämistä varten. Lisäksi solmun geometrian maksimivirhe esilasketaan. Katselijan etäisyyden, geometrian maksimivirheen, kuvaruudun koon ja kameran näkökentän (engl. *field of view*) perusteella voidaan laskea solmun maksimivirhe kuvaruudulla. Mikäli tämä virhe on enemmän kuin suurin sallittu virhe, solmu jaetaan. Pienin muutoksin menetelmä luultavasti soveltuisi myös pallomaisiin maastoihin.

Planeetan piirtoa varten tarvitaan vain nelipuiden lehtisolmut. Varsinaisia solmuhierarkioita ei siis luoda ollenkaan, vaan kustakin nelipuusta otetaan rekursion yhteydessä talteen vain lehtisolmut. Tämä tarkoittaa sitä, että nelipuut tulee käydä edellä esitetyllä tavalla läpi jokaisella kuvaruudun piirrolla. Nelipuiden läpikäynti on tosin niin nopeaa, ettei se osoittautunut pullonkaulaksi.

Kuten luvussa 4.2 nähtiin, 32-bittinen liukuluku ei riitä nelipuiden solmujen kulmapisteiden sijaintien esittämiseen tarkasti planetaarisella mittakaavalla. Siksi kulmapisteet esitetään 64-bittisillä liukuluvuilla. Nykyaikaiset näytönohjaimet kuitenkin käsittelevät paremmin 32-bittisiä liukulukuja. Piirtoa varten lehtisolmujen kulmapisteistä vähennetään 64-bittisillä liukuluvuilla esitetty kameran sijainti, jolloin kamerasta tulee pisteiden uusi origo. Nyt pisteet voidaan esittää 32-bittisillä liukuluvuilla, koska niissä on riittävästi tarkkuutta kameran lä-



Kuvio 14. Nelipuun jako. Kuvassa (a) on nelipuun juurisolmu, yksi alustavan kuution tahkoista. Kuvissa (b) – (d) nelipuu jaetaan tarkemmaksi punaisella pisteellä esitetyn katselijan lähestyessä planeettaa. Jokainen nelikulmio vastaa nelipuun lehtisolmua.

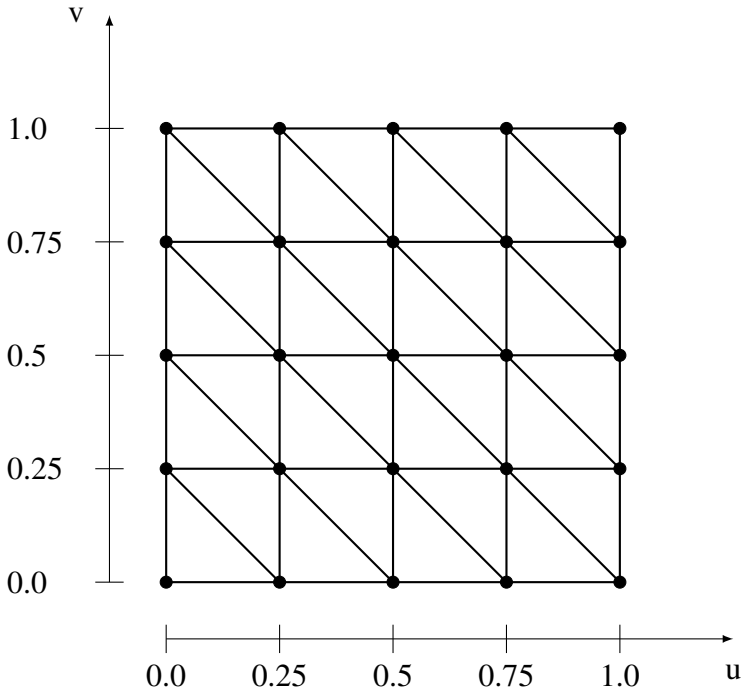
hellä oleville pisteille. Kaukana olevat pisteet eivät puolestaan tarvitse yhtä paljon tarkkuutta, koska virhe ei ole etäisyyden vuoksi havaittava.

5.1.2 Lehtisolmuista kolmioiksi

Näytönohjaimella piirrettävä kolmioverkko voitaisiin muodostaa lehtisolmujen kulmapisteiden välille, kuten Porwal (2013) tekee. Hän lisää jokaisen lehtisolmun keskelle pisteen ja muodostaa neljä kolmiota keskipisteen ja kulmapisteiden välille. Tällöin riittävän yksityiskohtaisen geometrian aikaansaamiseksi lehtisolmuja tarvitaan paljon, nelipuista tulee verrattain syviä ja näytönohjaimelle on lähetettävä suuri määrä kärkipistedataa jokaisella piirtokerralla. Kooima (2008) puolestaan muodostaa alustavan geometrian ROAM-algoritmin kaltaisella menetelmällä ja jakaa alustavat kolmiot useammiksi kolmioiksi näytönohjaimen laskentavarjostimella ennen piirtoa. Näin alustava kolmiointi saa olla karkea, ja siksi nopeas-

ti laskettavissa, koska näytönohjaimella lisättävät kolmiot takaavat riittävän yksityiskohtaisuuden.

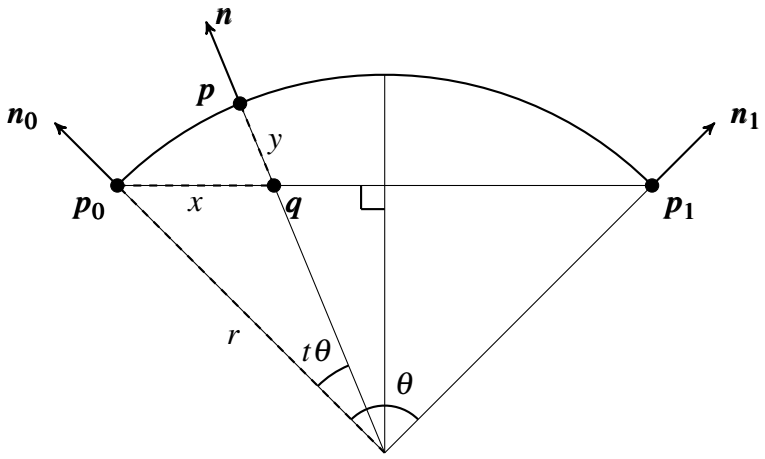
Tässä toteutuksessa nelipuiden lehtisolmut piirretään ruudukon mallisena kolmioverkkona, jossa on 32×32 kärkipistettä. Kolmioita puolestaan tulee $2 \cdot 31^2 = 1922$. Jokaisella kärkipisteellä on kaksi arvoa, $u, v \in [0,1]$. Ne kertovat, missä suhteessa piirrettävän lehtisolmun kulmapisteitä pitää yhdistää kyseisen kärkipisteen laskemiseksi. Näin ollen siis neljää kulmapistettä vastaavat (u,v)-parit ovat $(0,0)$, $(1,0)$, $(0,1)$ ja $(1,1)$. Kuviossa 15 on esitetty vastaava kolmioverkko, jossa kärkipisteitä on viisi sivua kohden. Kärkipisteiden sijainnit ja pallonormaalit lasketaan vasta näytönohjaimella kärkipistevarjostimessa, joten jokainen lehtisolmu voidaan piirtää käyttäen samoja kärkipiste- ja indeksipuskureita. Näytönohjaimelle tarvitsee viedä vain kulloinkin piirrettävän lehtisolmun kulmapisteet ja niiden pallonormaalit.



Kuvio 15. Parametrisoitu kolmioverkko. Kärkipisteiden u ja v -arvot kertovat, missä suhteessa lehtisolmun kulmapisteitä on yhdistettävä kärkipisteen laskemiseksi.

5.1.3 Kärkipisteiden sijainnit ja normaalit pallon pinnalla

Piirrettävän kolmioverkon kärkipisteiden sijainnit ja normaalit pallon pinnalla lasketaan kärkipistevarjostimessa. Kärkipisteiden sijainteja ei voida kuitenkaan laskea suoraan kulmapisteitä lineaarisesti interpoloimalla, koska sijainnit halutaan planeetan säteen etäisyydelle planeetan keskipisteestä. Toisaalta planeetan sädettä ei voida käyttää, koska se on liian suuri 32-bittisillä liukuluvuilla laskettaessa. Kuvio 16 havainnollistaa, miten laskut voidaan tehdä hyödyntäen trigonometriaa.



Kuvio 16. Kärkipisteiden sijaintien laskeminen käyttämättä sädettä.

Tavoitteena on laskea piste $\boldsymbol{p}$ ja sen pallonormaali $\boldsymbol{n}$ hyödyntäen pisteitä $\boldsymbol{p_0}$ ja $\boldsymbol{p_1}$ sekä normaaleja $\boldsymbol{n_0}$ ja $\boldsymbol{n_1}$. Normaali $\boldsymbol{n}$ on helppo laskea käyttäen pallomaista lineaarista interpolointia (engl. *spherical linear interpolation, SLERP*). Shoemake (1985) kertoo tästä interpolointimenetelmästä kvaternioihin sovellettuna [1], mutta se toimii myös kolmiulotteisten vektoreiden interpolointiin. Olkoon t välillä $[0,1]$ ja normaalien $\boldsymbol{n_0}$ ja $\boldsymbol{n_1}$ välinen kulma $\theta = \text{acos}(\boldsymbol{n_0} \cdot \boldsymbol{n_1})$. Pallomainen lineaarinen interpolointi antaa sellaisen normaalin $\boldsymbol{n}$, että normaalien $\boldsymbol{n_0}$ ja $\boldsymbol{n}$ välinen kulma on $t\theta$:

$$\boldsymbol{n} = \frac{\sin((1-t)\theta)}{\sin\theta}\boldsymbol{n_0} + \frac{\sin(t\theta)}{\sin\theta}\boldsymbol{n_1}$$

1. Tutkijan mielestä Jonathan Blow esittää pallomaisen lineaarisen interpoloinnin toimintaperiaatteen ymmärrettävämmin: `http://number-none.com/product/Understanding%20Slerp,%20Then%20Not%20Using%20It/` (luettu 25.4.2019).

Piste $\boldsymbol{p}$ puolestaan voidaan laskea seuraavasti:

$$\boldsymbol{p} = \boldsymbol{p_0} + x\frac{\boldsymbol{p_1} - \boldsymbol{p_0}}{\|\boldsymbol{p_1} - \boldsymbol{p_0}\|} + y\boldsymbol{n}$$

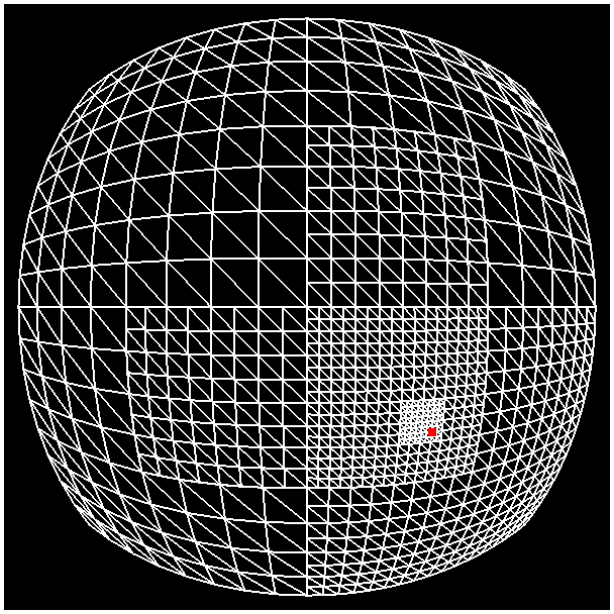
missä x kertoo, kuinka paljon pisteestä $\boldsymbol{p_0}$ pitää siirtyä pisteen $\boldsymbol{p_1}$ suuntaan, jotta päädytään pisteeseen $\boldsymbol{q}$, ja y, kuinka paljon pisteestä $\boldsymbol{q}$ pitää siirtyä normaalin $\boldsymbol{n}$ suuntaisesti, jotta päädytään pisteeseen $\boldsymbol{p}$. Luvut x ja y lasketaan seuraavasti:

$$x = (1 - \frac{\tan(\frac{\theta}{2} - t\theta)}{\tan\frac{\theta}{2}})\frac{\|\boldsymbol{p_1} - \boldsymbol{p_0}\|}{2}$$
$$y = (\frac{1}{\sin\frac{\theta}{2}} - \frac{1}{\cos(\frac{\theta}{2} - t\theta)\tan\frac{\theta}{2}})\frac{\|\boldsymbol{p_1} - \boldsymbol{p_0}\|}{2}$$

Jos $\boldsymbol{n_0}$ ja $\boldsymbol{n_1}$ ovat lähes yhdensuuntaiset, $\boldsymbol{p}$:n ja $\boldsymbol{n}$:n laskeminen tällä tavalla tuottaa virheellisiä tuloksia. Tämä tapahtuu, kun lehtisolmun kattama alue pallon pinnasta on lähes taso. Tällöin voidaan käyttää lineaarista interpolointia.

Tiettyä (u, v)-paria vastaavan $\boldsymbol{p}$:n ja $\boldsymbol{n}$:n laskeminen vaatii näiden laskujen suorittamista kolmesti: ensin kulmapisteiden $(0,0)$ ja $(1,0)$ sekä $(0,1)$ ja $(1,1)$ välillä u:n suuntaisesti ja sitten näistä laskuista saatujen pisteiden välillä vielä kerran v:n suuntaisesti. Kuviossa 17 on piirretty kuvion 14 d) esittämät lehtisolmut 5×5 kärkipisteen ruudukkoina edellä esitetyllä tavalla.

Esitetty tapa laskea kärkipisteiden sijainnit ja pallonormaalit on monimutkaisempi kuin Kooiman (2008) laskentavarjostinta hyödyntävässä menetelmässä. Hänen lähestymistapansa on periaatteeltaan rekursiivinen. Ensin alustavan kolmion sivujen keskipisteille lasketaan normaalit ja sijainnit. Näiden pisteiden kautta kolmio jaetaan neljäksi pienemmäksi kolmioksi, joille sama toistetaan. Laskutoimitusten yksinkertaisuus seuraa siitä, että uusi piste lasketaan aina kahden jo lasketun pisteen puoliväliin. Tällöin pallonormaalin laskeminen ei vaadi pallomaisen lineaarisen interpoloinnin käyttämistä, sillä normaali on kahden jo lasketun normaalin summavektorin suuntainen yksikkövektori. Sijainti puolestaan saadaan siirtämällä kahden muun pisteen keskipistettä pallonormaalin suuntaisesti pallon pinnalle. Kooima katsoo esilasketusta taulukosta, kuinka paljon pistettä tulee siirtää.



Kuvio 17. Yhden nelipuun lehtisolmut kolmioverkoilla piirrettynä. Punainen piste esittää katselijan sijaintia.

5.1.4 Korkeusarvojen ja normaalien generointi

Jokaiselle piirrettävälle lehtisolmulle luodaan 32×32-kokoinen nelikanavainen liukulukutekstuuri. Tekstuurissa on siis yksi pikseli jokaista lehtisolmun kolmioverkon kärkipistettä kohden. Kärkipisteiden korkeusarvot tallennetaan tekstuurin yhteen kanavaan ja normaalien x, y ja z -komponentit kolmeen muuhun. Lehtisolmua piirrettäessä korkeusarvot ja normaalit luetaan tekstuurista käyttäen tekstuurikoordinaatteina kärkipisteiden u ja v -arvoja. Tekstuurien sisältämien korkeusarvojen ja normaalien generoinnissa hyödynnetään erilaisia kolmiulotteisia Perlin-kohinoita. Jotta kohinat toimivat planetaarisen mittakaavan etäisyyksillä, niiden syötteenä saamat ja laskuissa käyttämät sijainnit esitetään 64-bittisillä liukuluvuilla.

Olkoon $\boldsymbol{p}$ kolmioverkon kärkipisteen sijainti pallon pinnalla. Kärkipisteen korkeusarvo saadaan näytteistämällä kohinaa pisteessä $\boldsymbol{p}$. Kärkipisteen normaali puolestaan saadaan seuraavasti:

- Lasketaan pisteen pallonormaalille kaksi tangenttivektoria $\boldsymbol{t}$ ja $\boldsymbol{u}$, jotka ovat keskenään kohtisuorassa ja yksikön mittaisia.
- Lasketaan lehtisolmun ja tekstuurin kokoon suhteutettu siirtovakio $s = \frac{l}{2k}$, missä l on

lehtisolmun sivujen pituuksien keskiarvo ja k on tekstuurin sivun pituus pikseleissä.

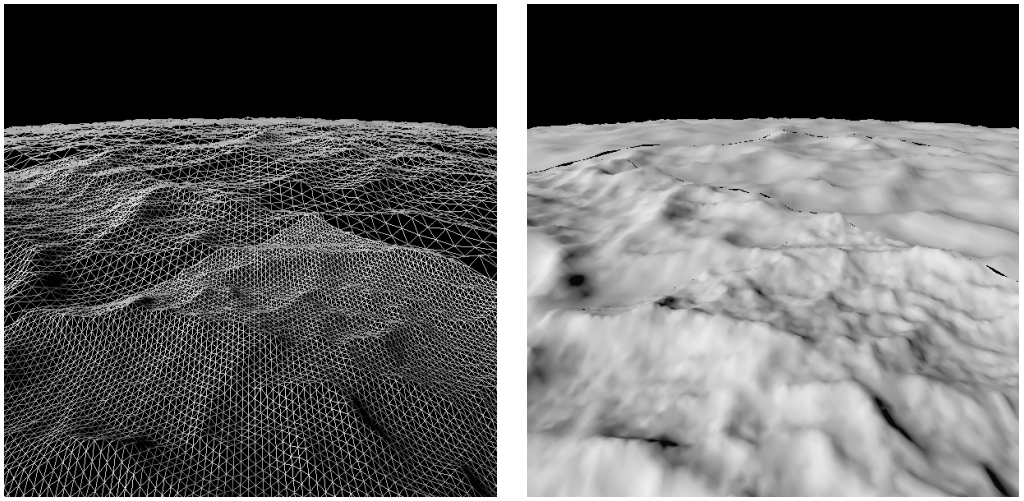
- Lasketaan neljä pistettä $\boldsymbol{a} = \boldsymbol{p} - s\boldsymbol{t}$, $\boldsymbol{b} = \boldsymbol{p} + s\boldsymbol{t}$, $\boldsymbol{c} = \boldsymbol{p} - s\boldsymbol{u}$ ja $\boldsymbol{d} = \boldsymbol{p} + s\boldsymbol{u}$ sekä viedään ne sitten pallon pinnalle.
- Näytteistetään kohinaa näissä pisteissä ja siirretään pisteitä niiden pallonormaalien suuntaisesti saatujen korkeusarvojen mukaan.
- Lasketaan vektorit $\boldsymbol{e} = \boldsymbol{b} - \boldsymbol{a}$ ja $\boldsymbol{f} = \boldsymbol{d} - \boldsymbol{c}$.
- Kärkipisteen normaali on vektorien $\boldsymbol{e}$ ja $\boldsymbol{f}$ välisen ristitulon suuntainen yksikkövektori.

Lehtisolmujen tekstuurit laitetaan talteen assosiaatiotauluun myöhempiä piirtokertoja varten. Assosiaatiotaulun avaimena toimii ID-luku, joka on yksilöllinen jokaiselle mahdolliselle nelipuun solmulle. ID on 64-bittinen. Ylintä bittiä käytetään validin ID-luvun tunnistamiseen: ID on validi vain, jos sen ylin bitti on 1. Kolme bittiä kertoo, mihin kuudesta nelipuusta solmu kuuluu ja viisi bittiä solmun syvyystason puussa. Jokaista syvyystasoa kohden tarvitaan vielä kaksi bittiä kertomaan, monesko kyseisen tason vanhempisolmun lapsista solmu on. Syvyystasoja voi olla enimmillään 27. Yksi 64:stä bitistä jää käyttämättä. Mikäli tallessa olevia tekstuureita on liikaa, tehdään uusille tilaa poistamalla tekstuuri, jonka edellisestä käytöstä on kulunut eniten aikaa.

Kuviossa 18 planeetan pinta on piirretty ensin rautalankamallina ja sitten täytetyillä kolmioilla. Kolmioiden kärkipisteitä on siirretty pallonormaalien suuntaisesti tekstuureista luettujen korkeusarvojen mukaisesti. Tekstuurit on luotu luvussa 3 esitetyn kaavan 2 mukaisella harjanteisella Perlin-kohinalla. Käytetty valaistusmalli on hyvin yksinkertainen. Heikkoon ambienttiin valoon summataan tekstuurista luetun normaalin ja valovektorin välinen pistetulo, jos se on positiivinen. Valovektori on jokaiselle pikselille sama, eli valonlähde on niin kutsuttu suunnattu valo (engl. *directional light*).

5.1.5 Raot ja helmojen käyttö

Jos vierekkäin olevat lehtisolmut ovat nelipuussa eri syvyydellä, niiden väliin voi jäädä rakoja. Ongelmaan on useita ratkaisuja. Esimerkiksi Porwal (2013) luo aluksi nelipuidensa jokaisen lehtisolmun kulma- ja keskipisteiden kautta neljä kolmiota. Sitten hän jakaa eri syvyystason solmujen rajalla suuremmat kolmiot samalle tasolle viereisten solmujen kolmioiden



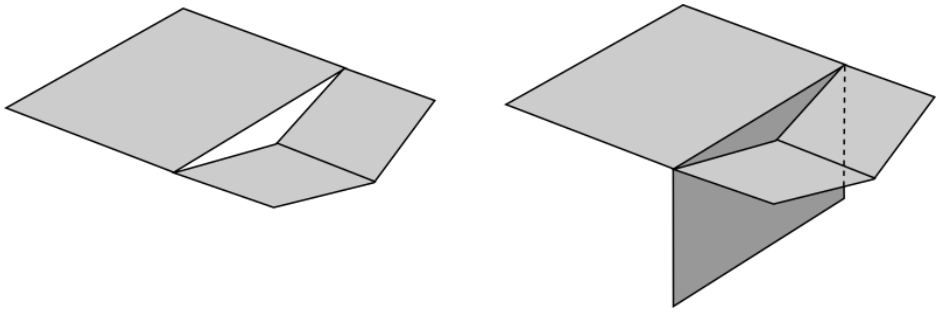
Kuvio 18. Planeetan pinta rautalankamallina sekä täytetyillä kolmioilla piirrettynä. Täytetyillä kolmioilla piirretyssä planeetan pinnassa on havaittavissa rakoja.

kanssa. Näin luotu kolmiointi on aukoton. Kooima (2008) sallii alustavan geometriansa vierekkäisten kolmioiden kokoluokan vaihdella enintään yhdellä ja piirtää pienemmän kolmion siten, että sen reuna yhdistyy saumattomasti viereiseen kolmioon.

Ulrich (2002) kuvailee erilaisia tapoja täyttää raot. Yksi niistä on helmojen (engl. *skirts*) käyttö. Helmojen toimintaperiaate on esitetty kuviossa 19. Helmat ovat osa lehtisolmun kolmioverkkoa. Reunojen kärkipisteisiin kiinnittyvät kaistaleet venytetään niin pitkälle maan alle kohti planeetan keskipistettä, että kaikki mahdolliset raot peittyvät. Helmat ovat yksinkertaisia, niitä on lähes mahdoton havaita, eivätkä ne vaadi mitään muutoksia toteutuksen muihin osiin, kuten rajoituksia viereisten lehtisolmujen syvyyseroille. Kuviossa 20 planeetan pinta on piirretty täytetyillä kolmioilla ilman helmoja (kuva a), jolloin raot näkyvät selvästi, sekä helmojen kanssa (kuva b), jolloin raot jäävät piiloon.

5.1.6 Z-kilpailu

Tavanomaisella syvyyspuskurilla Z-kilpailu oli selvästi havaittavissa etenkin kaukana olevassa maastossa ja kameran liikkuessa. Helmojen lisäyksen jälkeen tilanne paheni entisestään. Ongelman ratkaisu oli kuitenkin hämmästyttävän helppo. Syvyyspuskurista tehtiin logaritminen Kemenin (2012) esittämällä, luvussa 4.3 kuvaillulla tavalla. Kärkipistevarjosti-



Kuvio 19. Helmojen toimintaperiaate. Eri syvyystason solmujen välissä on rako (vas.), joka piilotetaan helmalla (oik.).

messa lasketaan logaritminen z-arvo, joka kirjoitetaan sitten pikselivarjostimessa syvyyspuskuriin. Toteutuksessa vakion c arvoksi valittiin 0.1. Näkymäpyramidin etuseinän etäisyys kamerasta on 1 metri ja takaseinän $200 \cdot 10^6$ metriä.

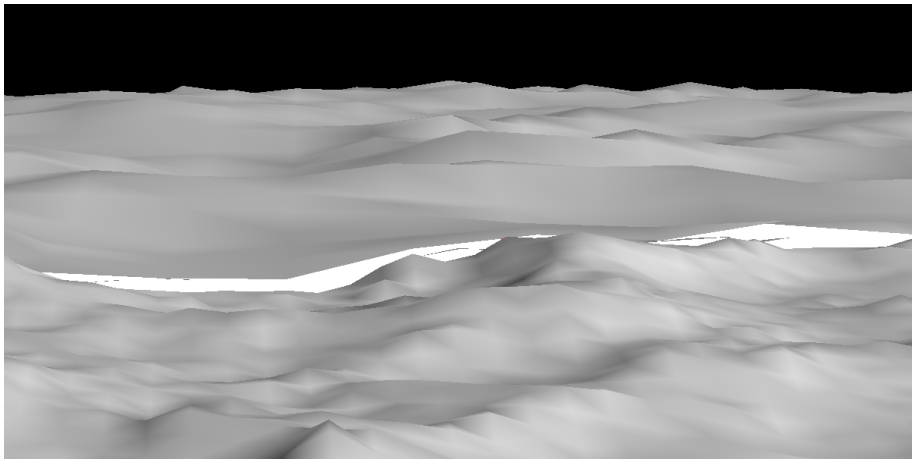
5.1.7 Reaaliaikaisuus

Korkeusarvoja ja normaaleja sisältävien tekstuurien generointi on raskasta. Jos kerralla joudutaan luomaan monta tällaista tekstuuria, menetelmä menettää reaaliaikaisuutensa. Tämän ratkaisemiseksi generoinnissa otettiin ensin käyttöön säikeistys: jokainen tekstuuria kaipaava lehtisolmu lisätään generointitehtävänä työjonoon, josta taustasäikeet ottavat tehtäviä suorittaakseen. Näin suorituskyky parani huomattavasti.

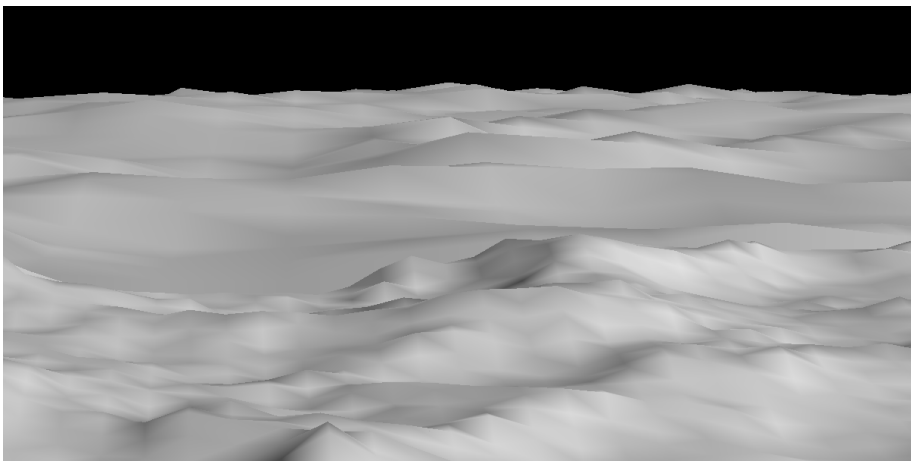
Säikeistetty generointi ei kuitenkaan yksistään riittänyt sulavaan käyttökokemukseen. Kameran liikkuessa suurella nopeudella nelipuiden rakenne voi muuttua valtavasti kuvaruutujen välillä. Tällöin kuvaruudun piirtämiseksi tarvitsee mahdollisesti generoida niin monta uutta tekstuuria, että säikeistetty generointikaan ei enää pysy perässä.

Menetelmään tehtiin merkittävä muutos, jonka ansiosta reaaliaikaisuus säilyy suuremmillakin nopeuksilla. Jos uusi lehtisolmu kaipaa korkeusarvojen ja normaalien generointia, tarkistetaan ensin, löytyykö jollekin sen vanhempisolmuista luotu tekstuuri. Jos tällainen löytyy, käytetään sitä solmun piirtämisessä, kunnes solmun oma tekstuuri on valmistunut. Vanhemman tekstuurista käytetään vain aluetta, jolla piirrettävän solmun korkeusarvot ovat. Vanhempisolmun tekstuuri ei toki ole yhtä yksityiskohtainen kuin solmun oma olisi, mutta ongelma

a)



b)



Kuvio 20. Solmujen väliin jäävät raot ja helmat. Kuvassa (a) eri syvyystason solmujen väliin jäävät raot näkyvät valkoisina. Kuvassa (b) raot on peitetty helmoilla.

kestää vain lyhyen hetken, kunnes solmun oma tekstuuri valmistuu. Tietenkin, jos minkään vanhempisolmun tekstuuria ei löydy, pitää solmun tekstuuri generoida ennen piirtämistä. Jos näin käy useille solmuille samalla piirtokerralla, näkyy se ikävänä "nykäyksenä"piirrossa. Kameran nopeuden pysyessä kohtuullisena näitäkään ei juuri ilmene.

Ylimääräiset grafiikkarajapinnan kautta tehtävät piirtokutsut ovat verrattain kalliita, vaikka yhtään monikulmiota ei lopulta piirrettäisikään. Menetelmän tehostamiseksi lehtisolmuille toteutettiin vielä näkymäpyramidilla leikkaus (engl. *frustum culling*), eli näkymäpyramidin ulkopuolella olevat solmut jätetään piirtämättä. Ennen solmun piirtämistä lasketaan, onko sille määritetty rajauspallo (engl. *bounding sphere*) näkymäpyramidin sisä- vai ulkopuolella. Jos rajauspallo on ulkopuolella, solmua ei piirretä. Lehtisolmun rajauspallo määritetään sille

aiemmin lasketun rajauslaatikon perusteella. Johtuen tavasta, jolla rajauslaatikko luotiin, se ei välttämättä sisällä kaikkia solmun pisteitä. Jotta solmua ei jätetä virheellisesti piirtämättä, rajauspallosta tehdään tarkoituksella monin verroin suurempi.

5.1.8 Maaston värjäys ja parempi kohinafunktion

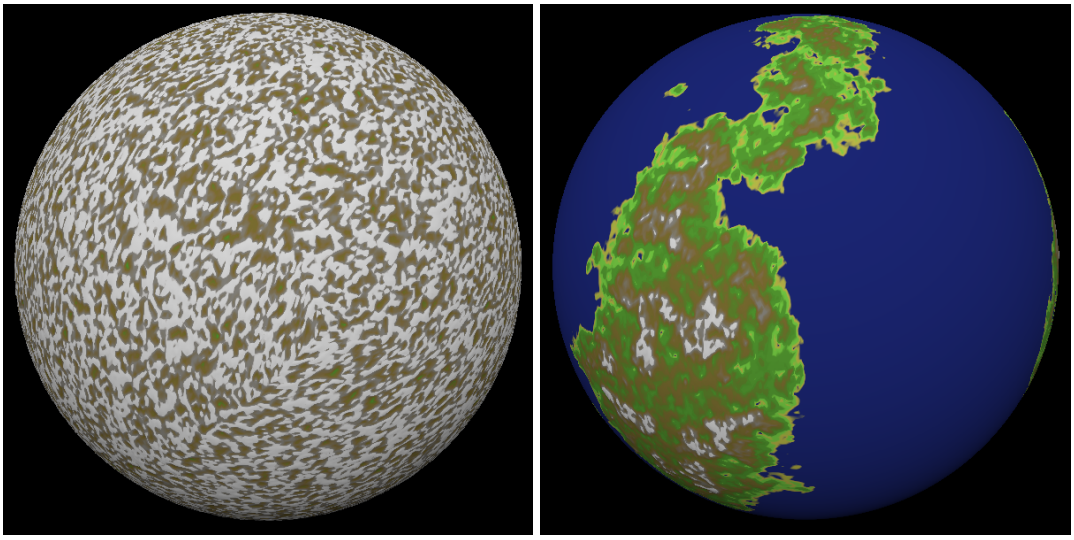
Kun menetelmän toteutus alkoi olla valmis, päätettiin maastoon lisätä väriä. Tämä tehtiin määrittämällä eri väriarvoja korkeuden mukaan. Värit paljastivat kuitenkin karun totuuden: käytetyn kohinafunktion tuottama maasto oli hyvin toistuvaa, epärealistista ja mielenkiinnotonta.

Maasto luotiin luvussa 3 esitetyn kaavan 2 mukaisella harjanteisella Perlin-kohinalla. Koska tämä ei miellyttänyt tutkijaa visuaalisesti, kehitettiin luvussa 3 esitetty yhdistelmäkohina. Myöhemmin tähän yhdistelmäkohinaan lisättiin vielä toista maskikohinaa käyttäen meret ja muut vesialueet.

Kuviossa 21 on ensimmäisenä planeetta, jonka maasto on luotu pelkällä harjanteisella Perlin-kohinalla ja värjätty yksinkertaisesti korkeuden mukaan. Kohinan yksipuolisuus on silmiinpistävää. Toisena kuviossa on mainitulla yhdistelmäkohinalla luotu planeetan maasto. Se on paljon mielenkiintoisempi ja vähemmän toistuva kuin edellinen, mutta sen generointi on tehottomampaa. Värjäys tehdään tässäkin korkeuden mukaan. Tutkielman liitteistä löytyy lisää kuvia tällä yhdistelmäkohinalla luodusta planeetan maastosta.

5.2 Vapaaseen kolmiointiin perustuva menetelmä

Nelipuumenetelmässä planeetan alustavan geometrian luonti aloitetaan aina kuutiosta. Tämä on luontevaa, koska nelipuita käytetään yleensä nelikulmaisten alueiden hierarkiseen jakoon, jollaisia kuution tahkot ovat. Tästä johtuen saman syvyystason lehtisolmujen kolmiot voivat olla huomattavan eri kokoisia ja muotoisia riippuen siitä, missä kohtaa planeettaa ne ovat. Lähellä kuution tahkon keskustaa kolmiot ovat lähes suorakulmaisia. Mitä kauemmaksi keskustasta mennään, sitä vähemmän kolmiot muistuttavat suorakulmaisia. Kaikkein epämuodostuneimpia kolmiot ovat kuution kulmapisteiden läheisyydessä.



Kuvio 21. Kaksi eri kohinafunktiolla generoitua pallomaista maastoa. Ensimmäisessä kuvassa kohinafunktio on pelkkä harjanteinen Perlin-kohina, toisessa on käytetty useamman kohinafunktion yhdistelmää.

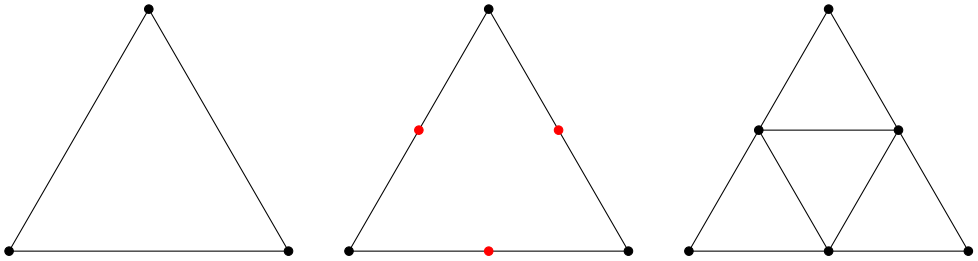
Jos algoritmi pohjautuisi nelikulmioiden jaon sijaan kolmioihin, kuten esimerkiksi Kooiman (2008) ROAM-algoritmiin perustuvassa menetelmässä, voitaisiin planeetan alustavan geometrian luonti aloittaa mistä tahansa monitahokkaasta, jonka tahkot on jaettu tarvittaessa ensin kolmioiksi. Esimerkiksi kuution tahkot voidaan kukin jakaa kahdeksi kolmioksi. Tällaista kolmioihin pohjautuvaa algoritmia käytettäessä alustava monitahokas voitaisiin valita siten, että lopullisista kolmioista tulisi mahdollisimman samanmuotoisia. Paras monitahokas tähän on säännöllinen ikosaedri (Kooima 2008).

Osana tutkielmaa toteutettiin tällainen kolmioihin perustuva menetelmä, joka pohjautuu nelipuita käyttävään toteutukseen. Menetelmä sai nimekseen vapaa kolmiointi, koska sen pohjalta kehitetyssä toisessa kolmioihin perustuvassa menetelmässä kolmiointi on tietyllä tapaa rajoitettu. Sen nimeksi tuli luonnollisesti rajoitettu kolmiointi.

5.2.1 Kolmioiden jako

Planeetan alustavan geometrian luonti aloitetaan monitahokkaasta, jonka tahkot ovat kolmioita tai jaettu kolmioiksi. Jokainen kolmio toimii oman puunsa juurisolmuna. Juurisolmut käydään vuorollaan läpi rekursiivisesti, kuten nelipuumenetelmässäkin. Jos solmu täyttää ja-

kokriteerin, se jaetaan neljäksi lapsisolmuksi, joille sama toistetaan. Jos jakokriteeri ei täyty, kyseessä on puun lehtisolmu, joka lisätään piirrettävien solmujen listaan. Kolmion jako lapsisolmuiksi on esitetty kuviossa 22. Ensin kolmion sivut puolitetaan ja sitten nämä uudet pisteet siirretään planeetan pinnalle. Lapsisolmut muodostetaan alkuperäisen kolmion kulmapisteiden ja uusien pisteiden välille. Jakokriteeri on sama kuin nelipuumenetelmässä, mutta koska solmut ovat kolmion muotoisia, niiden rajauslaatikot luodaan neljän kulmapisteen ja keskipisteen sijaan kolmesta kulmapisteestä ja keskipisteestä.



Kuvio 22. Kolmion jako lapsisolmuiksi.

Menetelmän lähtögeometriaksi valittiin Kelvinin rakenne. Se muodostuu kuudesta nelikulmiosta ja kahdeksasta kuusikulmiosta. Ennen alustavan geometrian luontia jokainen nelikulmio jaetaan kahdeksi kolmioksi ja jokaisen kuusikulmion keskelle lisätään piste, jonka kautta kuusikulmio jaetaan kuudeksi kolmioksi. Kaikkiaan juurikolmioita tulee siis $6 \cdot 2 + 8 \cdot 6 = 60$.

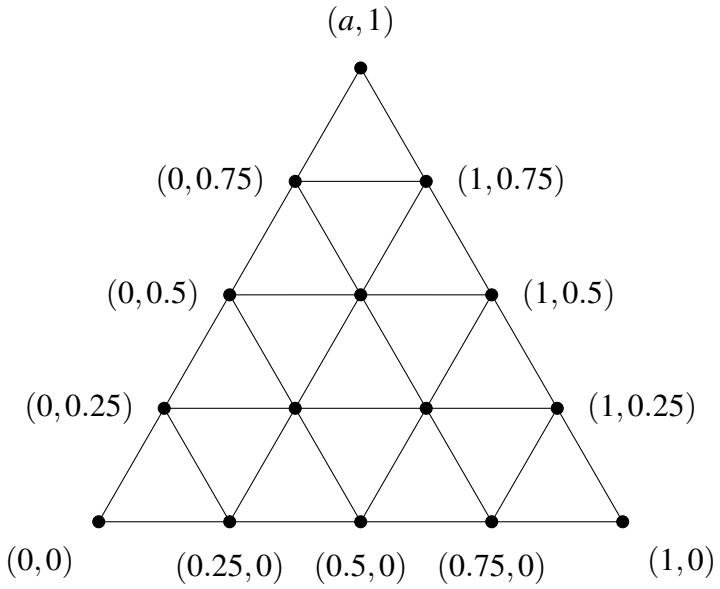
5.2.2 Lehtisolmujen piirto

Kuten nelipuumenetelmässä, jokainen lehtisolmu piirretään tiheämpänä kolmioverkkona. Kolmioverkon kärkipisteillä on vastaavasti kaksi arvoa $u, v \in [0,1]$, joiden mukaan kulloisenkin solmun kulmapisteitä yhdistetään. Näytönohjaimelle viedään vain solmun kulmapisteiden sijainnit (origona kameran sijainti) ja pallonormaalit. Kaikki lehtisolmut voivat siis jakaa saman kolmioverkon kärkipiste- ja indeksipuskurit.

Kolmioverkko on periaattessa saatu jakamalla yksi kolmio samaan tapaan kuin edellä esitetyssä alustavan geometrian luonnissa. Tästä saaduille neljälle pienemmälle kolmiolle tehdään sama jako-operaatio. Taas jako toistetaan uusille pienemmille kolmioille. Kun jako-operaatio on toistettu näin n kertaa, kolmioita on 4^n kappaletta. Alkuperäisen kolmion sivua

kohden puolestaan on $2^n + 1$ kärkipistettä. Toteutuksessa $n = 5$, eli kolmioita tulee $4^5 = 1024$ ja kärkipisteitä sivua kohden $2^5 + 1 = 33$.

Kärkipisteiden u ja v toimivat hieman erikoisesti. Tämä johtuu tavasta, jolla solmun kulmapisteitä yhdistetään kärkipisteiden sijaintien ja pallonormaalien laskemiseksi. Kuvio 23 havainnollistaa kolmioverkkoa ja sen eri kärkipisteiden (u,v)-pareja kolmiolla, jolle edellä esitetty jako-operaatio on tehty kahdesti. Olkoon solmun kulmapisteet A, B ja C sellaiset, että pistettä A vastaa (u,v)-pari $(0,0)$, pistettä B $(1,0)$ ja pistettä C kaikki (u,v)-parit $(a,1)$, missä $a \in [0,1]$. Jos halutaan laskea esimerkiksi kärkipiste $(0.5,0.5)$, lasketaan ensin kaksi pistettä yhdistämällä kulmapisteitä A ja C sekä B ja C v:n osoittamassa suhteessa. Saadaan pisteet $(0,0.5)$ ja $(1,0.5)$. Tämän jälkeen nämä kaksi pistettä yhdistetään u:n osoittamassa suhteessa lopullisen pisteen saamiseksi. Siis v:n kasvaessa lähestytään kulmapistettä C. Kun $v = 1$, ollaan kulmapisteessä C, olipa u mikä tahansa. Toteutuksessa kulmapisteelle C valittiin $u = 0$.



Kuvio 23. Kolmioverkko ja kärkipisteiden (u,v)-parit. Lehtisolmun kulmapisteitä yhdistetään kärkipisteen u ja v arvojen osoittamassa suhteessa kärkipisteen sijainnin ja pallonormaalin laskemiseksi.

Kahden pisteen yhdistäminen suhteessa $t \in [0,1]$ tapahtuu samalla tapaa käyttäen trigonometriaa kuin nelipuumenetelmässä. Lehtisolmuille piirretään myös helmat samoin kuin ne-

lipuumenetelmässä.

5.2.3 Korkeus- ja normaalikarttojen generointi

Solmun korkeus- ja normaaliarvot sisältävä tekstuuri luodaan hieman toisella tapaa kuin nelipuumenetelmässä. Tekstuuri on saman kokoinen 32×32 nelikanavainen liukulukutekstuuri, joka sisältää vastaavasti korkeusarvot yhdessä kanavassa ja normaalien x, y ja z -komponentit kolmessa muussa. Ero piilee siinä, ettei tekstuurissa ole jokaista solmun kolmioverkon kärkipistettä kohden tarkalleen yhtä pikseliä.

Tekstuuria generoitaessa jokainen pikseli käydään vuorollaan läpi. Olkoon $n = 32$ tekstuurin pikselisarakkeiden määrä ja $i \in \mathbb{Z}$ kulloisenkin pikselin sarakkeen indeksi, $0 \leq i < n$. Vastaavasti olkoon $m = 32$ tekstuurin pikselirivien määrä ja $j \in \mathbb{Z}$ pikselin rivin indeksi, $0 \leq j < m$. Pikseliä vastaavat $u, v \in [0,1]$ arvot ovat $u = \frac{i}{n-1}$ ja $v = \frac{j}{m-1}$. Näitä u ja v arvoja vastaava piste lasketaan yhdistämällä solmun kulmapisteiden sijainteja ja pallonormaaleja samaan tapaan kuin solmua piirrettäessä. Tälle pisteelle generoidaan korkeusarvo ja normaali, kuten nelipuumenetelmässä, ja saadut arvot tallennetaan pikseliin. Kolmioverkkoa piirrettäessä kärkipisteiden u ja v -arvot toimivat tekstuurikoordinaatteina korkeusarvoja ja normaaleja luettaessa.

Näin luotu tekstuuri ei ole optimaalinen. Itse asiassa lähes puolet tekstuurista on käytetty turhaan. Kolmioverkon 33 kärkipistettä, joiden $v = 0$, saavat korkeusarvonsa ja normaalinsa tekstuurin ensimmäiseltä riviltä $j = 0$, jolla on vain 32 pikseliä. Toisaalta viimeisellä rivillä $j = 31 \implies v = 1$, joten rivin jokaisen pikselin korkeusarvo ja normaali on generoitu samassa pisteessä, joka on solmun kulmapiste C, kuten edellisessä aliluvussa esitettiin.

Koska tekstuurit toimivat näin kummallisesti, solmuja piirrettäessä ei voida hyödyntää vanhempisolmujen tekstuureja samalla tapaa kuin nelipuumenetelmässä. Tämä onnistuisi esimerkiksi, jos tekstuurista käytettäisiin kolmion muotoinen alue, jolle solmun kärkipisteet kuvautuisivat yksiselitteisesti. Tällöin lapsisolmun kattava alue olisi helppo löytää tekstuurista. Jos tämä saataisiin toimimaan, menetelmä olisi luultavasti reaaliaikainen myös kameran liikkuessa nopeasti. Myös tekstuurit voitaisiin mahdollisesti jakaa kahden solmun kesken, jolloin tekstuurimuistin tarve puolittuisi. Koska esitettyä parannusta ei ehditty toteuttaa,

menetelmä ei ole ollenkaan sulava, jos kamera liikkuu niin nopeasti, että alustavan geometrian kolmiointi muuttuu huomattavasti ja useita uusien lehtisolmujen tekstuureita joudutaan generoimaan kuvaruudun piirtoa varten.

5.2.4 Solmujen ID-luvut

Kuten nelipuumenetelmässä, solmujen tekstuurit tallennetaan assosiaatiotauluun, jonka avaimena toimii solmun ID-luku. ID-lukua piti hieman muuttaa, jotta se soveltui tähän toteutukseen. Kolme bittiä ei riitä juurisolmuille, jos niitä on enemmän kuin kahdeksan. Esimerkiksi aiemmin esitetyllä tavalla kolmioiksi jaettu Kelvinin rakenne tarvitsee 60 juurisolmua. Syvyystasojen enimmäismäärä laskettiin 26:een, jolloin juurisolmuille saatiin kaksi bittiä lisää. Vielä yksi käyttämätön bitti otettiin juurisolmujen esittämiseen. Näin ollen ID-luvun kaikki bitit ovat käytössä, joista kuusi kertoo juurisolmun. Juurisolmuja voi siis olla enimmillään $2^6 = 64$.

5.3 Rajoitettuun kolmiointiin perustuva menetelmä

Sekä nelipuumenetelmässä että vapaassa kolmioinnissa on toisinaan melko selvästi havaittava sauma kahden eri syvyystason solmun välillä. Jos solmujen reunat saataisiin piirrettyä sopivalla kolmioinnilla siten, että rajasta tulisi tiivis, enimpien saumojen pitäisi kadota. Jos naapurisolmut yhdistyvät tiiviisti, myös helmat voidaan unohtaa.

Kolmas menetelmä, rajoitettu kolmiointi, on toteutukseltaan lähes täysin samanlainen kuin vapaa kolmiointi. Ainoat erot ovat alustavan geometrian luonnissa ja lehtisolmujen piirrossa. Vapaan kolmioinnin alustavassa kolmioinnissa solmuja pilkotaan vapaasti vaikuttamatta naapurisolmujen syvyystasoon. Rajoitetussa kolmioinnissa naapurisolmujen syvyystasojen ero rajoitetaan yhteen. Tarkoituksena on piirtää solmut siten, että naapurisolmut yhdistyvät saumoitta toisiinsa. Solmua piirrettäessä naapurisolmujen syvyystasojen perusteella valitaan kolmioverkko, jonka reunat sopivat yhteen naapureiden kanssa. Tällä tavoin myös Kooima (2008) piirtää alustavat kolmionsa.

5.3.1 Alustava kolmiointi

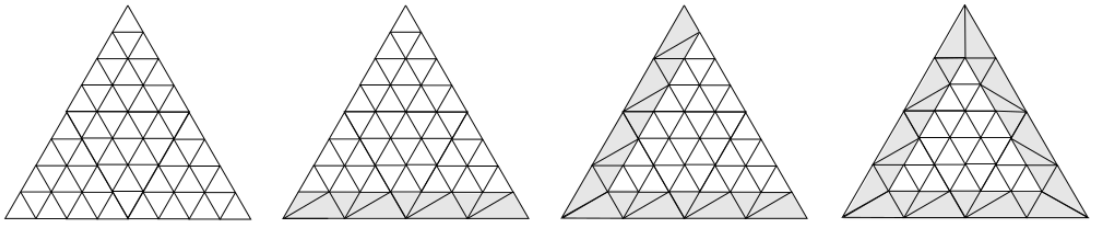
Rajoitettu kolmiointi aloitetaan vapaan kolmioinnin tavoin jostain monitahokkaasta, jonka tahkot ovat kolmioita tai tarvittaessa jaettu kolmioiksi. Kaikki nämä kolmiot toimiva juurisolmuina, ja ne lisätään tyhjään solmulistaan. Jokaisella solmulla on tieto siihen kolmen sivunsa kautta yhdistyvistä naapureista. Solmulistaa ruvetaan käymään järjestyksessä läpi solmu kerrallaan käyttäen samaa jakokriteeriä kuin vapaassa kolmioinnissakin.

Jos solmu on tarpeen jakaa, ensin tarkistetaan, seuraisiko jaosta naapureihin verrattuna suurempi kuin yhden syvyystason ero. Jos solmut ovat vielä jaonkin jälkeen enimmillään yhden syvyystason etäisyydellä toisistaan, tarkasteltavana oleva solmu jaetaan neljäksi lapsisolmuksi, kuten vapaassa kolmioinnissa. Sekä naapureiden että uusien lapsisolmujen naapuruustiedot päivitetään. Uudet solmut lisätään solmulistaan, jotta ne käydään vastaavasti vuorollaan läpi ja tarvittaessa pilkotaan taas pienemmiksi. Jos solmun jaosta seuraisi liian suuri syvyysero jonkin naapurisolmun kanssa, samanlainen jakotoimenpide suoritetaan ensin kyseiselle naapurisolmulle, mistä voi rekursiivisesti seurata edelleen lisää solmujen jakoja. Toimintaperiaate on samankaltainen kuin ROAM-algoritmissa (Duchaineau ym. 1997).

5.3.2 Lehtisolmujen piirto

Kun kaikki solmut on käyty läpi edellä esitetyllä tavalla, on aika piirtää lehtisolmut. Solmulista käydään läpi ja piirrettäväksi valitaan ne solmut, joilla ei ole lapsia. Näiden on oltava lehtisolmuja. Kunkin lehtisolmun naapurisolmujen syvyystasojen mukaan määräytyy, minkälainen kolmioverkko solmun piirtoon tarvitaan. Toteutuksessa käytetään kaikkiaan kahdeksaa erilaista kolmioverkkoa. Kuviossa 24 on esitetty niistä neljä. Kahdesta keskimmäisestä on molemmista vielä kaksi eri suuntaan pyöräytettyä versiota.

Jos piirrettävä lehtisolmu on samalla tai korkeammalla syvyystasolla kaikkien sen naapurisolmujen kanssa, käytetään tavallista, kuviossa vasemmanpuolimmaisena esitettyä kolmioverkkoa. Siitä seuraavaa käytetään, jos naapurisolmuista yksi on korkeamman tason solmu. Taas siitä seuraavaa käytetään, jos naapurisolmuista kaksi on korkeamman tason solmuja. Viimeistä käytetään, jos kaikki naapurit ovat korkeamman tason solmuja. Koska suurin sallittu syvyysero on yksi, näillä kolmioverkoilla pitäisi saada kaikki naapurisolmut yhdistettyä.



Kuvio 24. Neljä mahdollista kolmioverkkoa. Harmaalla korostetut reunat yhdistyvät korkeamman syvyystason solmuun.

Kun menetelmä oli toteutettu, havaittiin kuitenkin, että solmut eivät yhdisty saumoitta toisiinsa. Myöskään yhtymäkohta ei ole tiivis. Helmoista ei voidakaan siis luopua. Kuviossa 25 on esitetty kahden eri syvyystason solmun raja. Ensimmäisen kuvan rautalankamallista nähdään, kuinka solmujen kolmioverkot eivät aivan täsmää rajalla. Keskimmäisessä kuvassa tämä näkyy ilmeisenä rakona, kun helmat ovat poissa käytöstä. Viimeisessä kuvassa nähdään sauma siinä kohtaa, missä helma peittää raon.

Kuvio 25. Kahden naapurisolmun raja. Ensimmäisessä kuvassa nähdään, kuinka tiheämpi kolmioverkko yhdistyy väljempään, toisessa havaitaan rako solmujen välillä ja kolmannessa helmalla peitetyn raon paikkeilla erottuu sauma.

Herää kysymys, miksi reunat eivät täsmää? Ainakin yksi syy lienee se, että kahden eri syvyystason solmun tekstuurit eivät ole yhtenevät solmujen rajalla. Niitä on myös hankala saada täsmäämään nykyisellä toteutuksella, jossa jokaiselle kolmioverkon kärkipisteelle ei ole omaa pikseliä solmujen tekstuureissa.

Vaikka reunoja ei saatu yhdistymään saumoitta, eikä helmoista päästy eroon, menetelmä ei

silti välttämättä ole täysi susi. Jo pelkästään naapurisolmujen syvyyseron rajoittaminen vähentää selkeimpiä saumakohtia maastossa. Lisäksi reunoistaan paremmin yhtyvät kolmioverkot näyttävät myös auttavan hieman. Jää kuitenkin pohdittavaksi, onko menetelmän monimutkaisuus siitä saatavan, odotettua pienemmän hyödyn arvoista.

6 Kolmen menetelmän vertailu

Seuraavaksi kolmea toteutettua menetelmää verrataan keskenään muutaman eri ominaisuuden valossa. Koska osa vertailukohdista on enemmän tai vähemmän subjektiivisia, vertailu on näiden kohdalla ensisijaisesti tutkijan näkemys menetelmien paremmuusjärjestyksestä eikä niinkään universaali totuus.

Ensin vertaillaan menetelmien hankalasti mitattavia, enimmäkseen subjektiivisia ominaisuuksia. Tämän jälkeen esitetään testiasetelma -ja laitteisto, joilla menetelmiä mitattiin objektiivisemmin. Sitten esitetään mittauksen tulokset ja tehdään niistä joitain johtopäätöksiä. Lopuksi menetelmien vertailu vedetään yhteen.

6.1 Menetelmien arviointi

Nelipuumenetelmä on yksinkertaisin toteuttaa. Moni asia, mikä on helppo ajatella ja toteuttaa nelikulmioina, on kolmioilla paljon hankalampaa. Esimerkiksi kärkipisteen laskeminen neljästä kulmapisteestä on triviaalia, mutta kolmella kulmapisteellä tilanne ei olekaan niin yksinkertainen. Nelikulmainen kolmioverkko istuu myös hyvin nelikulmaiseen tekstuuriin. Kolmio on hankalaa sovittaa tekstuuriin siten, että jokaista kärkipistettä kohden löytyy yksi pikseli. Rajoitettu kolmiointi on menetelmistä monimutkaisin, koska se perustuu kolmioihin ja lisäksi alustavan geometrian luonnissa sekä solmujen piirrossa tulee huomioida naapurisolmut.

Nelipuumenetelmän pohjalla on kuutio. Kuution kulmapisteiden lähellä piirrettävät kolmiot ovat hyvin vääristyneitä. Kolmiointiin perustuvat menetelmät puolestaan toimivat millä tahansa monitahokkaalla [1]. Oikein valittu monitahokas johtaa vain vähäiseen kolmioiden vääristymiseen. Siksi kolmiointimenetelmät ovat visuaalisesti miellyttävämpiä kuin nelipuumenetelmä. Rajoitetussa kolmioinnissa solmut yhdistyvät kaikkein miellyttävimmin. Se ei kuitenkaan toimi aivan kuten oli suunniteltu, joten ero vapaaseen kolmiointiin on melko vähäinen.

1. Toteutettu ID-luku tosin asettaa juurisolmuina toimivien kolmioiden määrälle rajan 64, mutta tarpeen vaatiessa rajaa voidaan kasvattaa.

6.2 Testiasetelma ja -laitteisto

Menetelmiä ei ole mielekästä testata ja verrata kameran ollessa liikkeessä, koska kumpikaan kolmiointimenetelmistä ei hyödynnä vanhempisolmujen korkeus- ja normaalikarttoja maaston generoinnin aiheuttaman viiveen piilottamiseksi. Juuri tämän ominaisuuden ansiosta nelipuumenetelmä toimii reaaliaikaisesti nopeammassakin liikkeessä. Sen sijaan kameran ollessa paikallaan tästä ominaisuudesta ei ole nelipuumenetelmälle hyötyä, kunhan kaikkien piirrettävien solmujen kartat on saatu generoitua.

Jokaista menetelmää testattiin kahdessakymmenessä eri pisteessä. Kauimmaisen pisteen etäisyys planeetan pinnasta oli 10^7 metriä, seuraavaksi kauimmaisen puolet tästä, seuraavan taas puolet edellisestä, ja niin edelleen. Lähimmän pisteen etäisyys planeetan pinnasta oli siis $\frac{10^7}{2^{19}} \approx 19$ metriä. Kamera asetettiin vuorollaan jokaiseen pisteeseen siten, että horisontti näkyi kuvaruudulla. Horisontin näkymisen arveltiin vastaavan tavanomaista tilannetta reealiaikaisissa sovelluksissa, kuten peleissä.

Kamera pidettiin paikallaan koko kulloisessakin pisteessä tehdyn mittauksen ajan. Ennen varsinaista mittausta odotettiin niin kauan, että kaikki generointia odottavat tekstuurit olivat varmasti valmiita, jottei generointi vain sotkenut mittauksia. Tämän jälkeen mitattiin kymmenen sekunnin ajalta keskimääräinen kuvaruudun piirtoon kulunut aika. Myös kuvaruudun piirtoon tarvittavien grafiikkarajapinnan kautta tehtyjen piirtokutsujen sekä vairsinaisten piirrettyjen kolmioiden määrät laskettiin.

Lisäksi korkeus- ja normaalikarttojen tekstuurimuistintarve m arvioitiin kaavalla $m = pwhn$, missä p on yhden pikselin vaatima tavumäärä, w ja h ovat tekstuurin leveys ja korkeus pikseleissä sekä n on tekstuurien lukumäärä. Jokaisessa menetelmässä tekstuurit ovat saman kokoisia neliön muotoisia nelikanavaisia liukulukutekstuureja. Siis yksi pikseli vaatii neljä nelitavuista liukulukua, joten $p = 16$ tavua. Tekstuurien koko on puolestaan $w = h = 32$. Näin ollen yhden tekstuurin pikselidata vaatii $pwh = 16 \cdot 32 \cdot 32 = 16384$ tavua, eli 16 kilotavua. Siis kaikissa menetelmissä $m = 16n$ kilotavua.

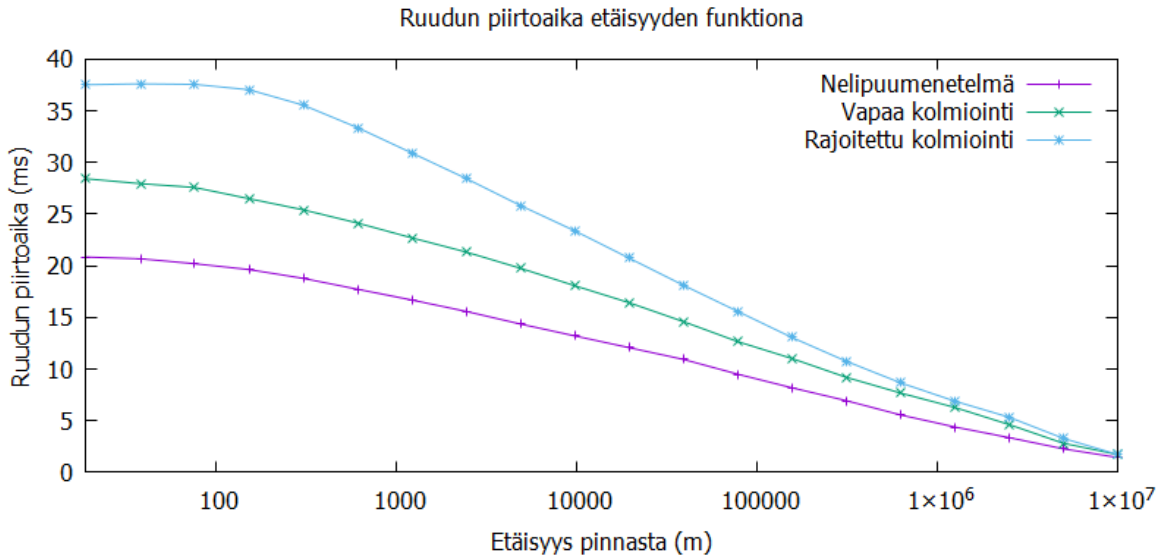
Testaus suoritettiin kannettavalla tietokoneella, jonka oleellisimmat tekniset tiedot ovat seuraavat:

- Näytönohjain: NVIDIA GeForce GT 550M, jossa on 2 gigatavua DDR3 muistia.
- Prosessori: Intel Core i7-2670QM, jossa on neljä varsinaista ydintä, mutta Intelin hypersäikeistys -teknologialla varustettuna jokainen ydin toimii kahtena loogisena ytimenä. Kaikkien kahdeksan loogisen ytimen kellotaajuus on 2.20 gigahertsiä.
- Keskusmuisti: 14 gigatavua DDR3 muistia.
- Käyttöjärjestelmä: Windows 7 Professional N, 64-bit.

Testauksessa käytetty kuvaruudun resoluutio oli 1280×720.

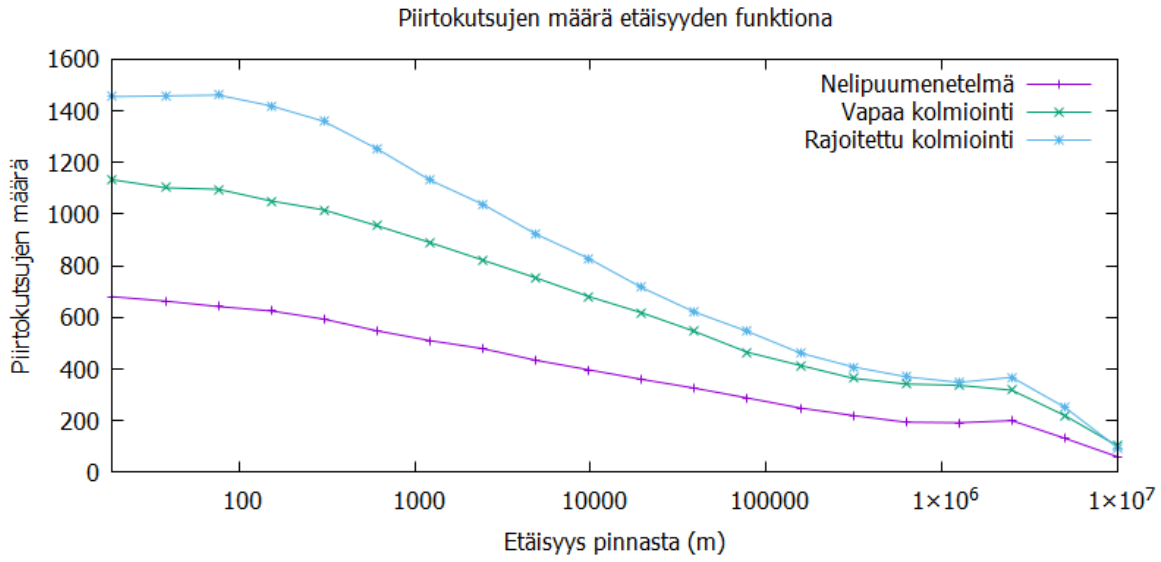
6.3 Mittaustulokset ja johtopäätöksiä

Jokaisen menetelmän keskimääräinen yhden kuvaruudun piirtoon kulunut aika eri etäisyyksillä on esitetty kuviossa 26. Kuvio 27 puolestaan esittää, kuinka monta piirtokutsua kukin menetelmä tekee kuvaruudun piirron aikana eri etäisyyksellä. Kuten kuvaajista näkyy, nelipuumenetelmä tekee vähiten piirtokutsuja ja on suorituskykyisin kaikilla etäisyyksillä.



Kuvio 26. Ruudun piirtoon kulunut aika eri etäisyyksillä planeetan pinnasta.

Yksi selittävä tekijä ainakin rajoitetun kolmioinnin heikompaan kuvaruudun piirtoaikaan lienee monimutkaisempi alustavan geometrian luonti. Jotta naapurisolmujen välillä on aina korkeintaan yhden syvyystason ero, solmuja joudutaan jakamaan toisinaan pienemmiksi, kuin



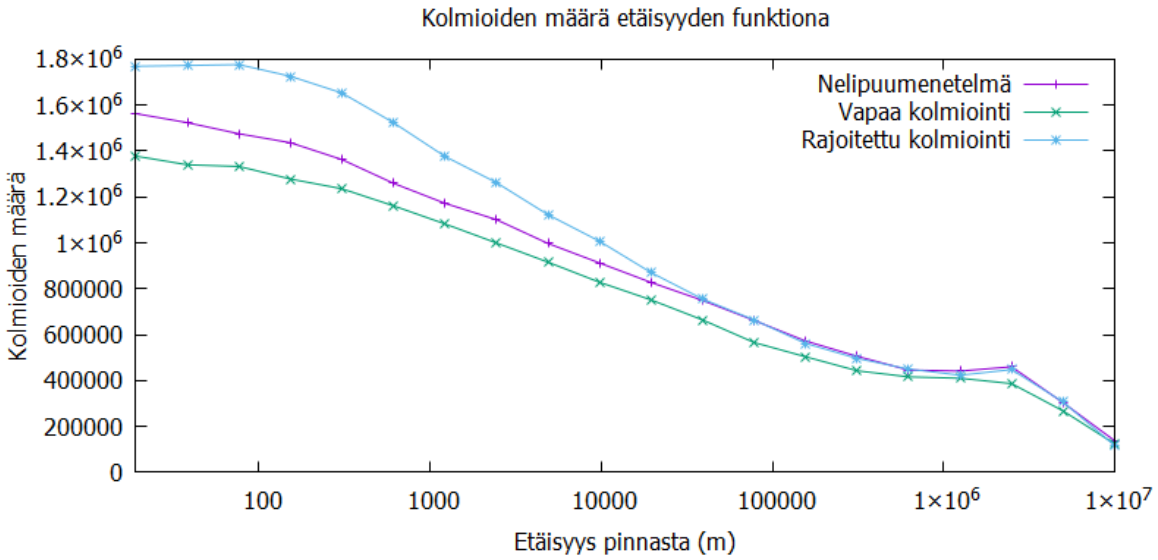
Kuvio 27. Piirtokutsujen määrä eri etäisyyksillä planeetan pinnasta.

muutoin olisi ollut tarpeen. Tällöin myös piirrettävää tulee enemmän. Myös vapaassa kolmioinnissa piirtokutsuja tarvitaan enemmän kuin nelipuumenetelmässä, koska yksi kolmion muotoinen lehtisolmu kattaa oletettavasti keskimäärin pienemmän alueen planeetan pinnasta kuin neliskulmainen.

Grafiikkarajapinnan kautta tehtävien piirtokutsujen eriävät määrät saattavat selittää suuren osan menetelmien välisistä eroista suorituskyvyssä. Tämän testaamiseksi voitaisiin toteuttaa instansointina tunnettu menetelmä: yhdellä kutsulla piirretään useita eri instansseja kolmiulotteisesta kappaleesta, tässä tapauksessa lehtisolmuja. Mittaus pitäisi suorittaa uudelleen instansoinnin kanssa ja verrata tuloksia aiempaan mittaukseen. Oletettavasti jokainen menetelmistä hyötyisi jossain määrin instansoinnista.

Kuvio 28 esittää yhden kuvaruudun aikana piirrettyjen kolmioiden määrän eri etäisyyksillä planeetan pinnasta. Nähdään, että rajoitettu kolmiointi piirtää lähes poikkeuksetta selvästi enemmän kolmioita kuin kaksi muuta menetelmää. Tämä on odotettua, koska rajoitus naapurisolmujen syvyyseroille aiheuttaa enemmän piirrettävää, mikä näkyi myös kuviossa 27. Sen sijaan nelipuumenetelmä piirtää enemmän kolmioita, kuin vapaa kolmiointi. Siitä huolimatta nelipuumenetelmä on menetelmistä tehokkain kuvaruudun piirtoajassa. Tämä puoltaa ajatusta, että kolmiointimenetelmien hitaus ainakin osin seuraisi piirtokutsujen määrästä.

Piirrettävien kolmioiden määrä ei tunnu sitä selittävän.



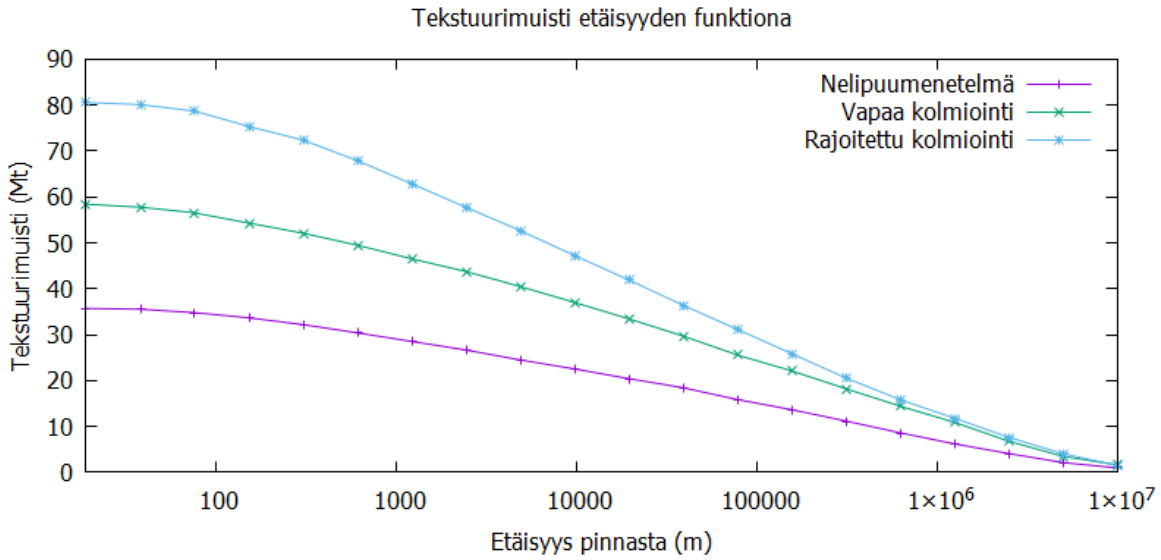
Kuvio 28. Piirrettyjen kolmioiden määrä eri etäisyyksillä planeetan pinnasta.

Menetelmien vaatima arvioitu tekstuurimuistin määrä eri etäisyyksillä planeetan pinnasta on esitetty kuviossa 29. Jälleen nelipuumenetelmä näyttää olevan tehokkain. Kuitenkin on pidettävä mielessä, että kolmiointimenetelmissä yhdestä tekstuurista on käytetty turhaan lähes puolet. Jos kaksi solmua voisi jakaa yhden tekstuurin, kuten aiemmin esitettiin, molemman kolmiointimenetelmän tekstuurimuistintarve puolittuisi. Tällöin vapaa kolmiointi olisi luultavasti vähintään yhtä tehokas tai jopa tehokkaampi muistin käytössä kuin nelipuumenetelmä. Rajoitettu kolmiointikaan ei jäisi paljon jälkeen.

Mikään menetelmistä ei ole varsinaisesti muistisyöppö. Rajoitetun kolmioinninkin maksimi muistintarve, hieman yli 80 megatavua on varsin vähän, jos vertaa käytettävissä olevaan muistiin. Tämän perusteella jokainen menetelmistä kelpaisi luultavasti useimpiin käyttötarkoituksiin.

6.4 Vertailun yhteenveto

Kolmesta menetelmästä parhaaksi arvioitiin nelipuumenetelmä. Se on yksinkertaisin toteuttaa, tehokkain muistin käytön ja piirtoajan osalta sekä raa'assa kolmioiden piirrossa. Lisäksi



Kuvio 29. Tekstuurimuistin tarve eri etäisyyksillä planeetan pinnasta.

se toimii reaaliaikaisesti myös kameran liikkuessa nopeasti.

Kolmiointiin perustuvien menetelmien suurimpana etuna on joustavuus lähtögeometrian valinnassa, ja oikein valittuna saman syvyystason solmujen piirrettävistä kolmioista tulee lähes yhdenmuotoisia ja -kokoisia (Kooima 2008). Rajoitettu kolmiointi on menetelmistä visuaalisesti miellyttävin naapurisolmujen ollessa korkeintaan yhden syvyystason päässä toisistaan ja solmujen kolmioverkkojen rajojen vastatessa paremmin toisiaan. Se on kuitenkin hankalin toteuttaa, eikä menetelmän hyödyt ole kovin merkittävät vapaaseen kolmiointiin verrattaessa.

Vapaa kolmiointi voitaisiin luultavasti kohtuullisella vaivalla parannella suorituskyvyssä lähes samalle tasolle nelipuumenetelmän kanssa. Muistin käytössä vapaa kolmiointi voisi jopa alittaa nelipuumenetelmän, mikäli yksittäinen tekstuuri saataisiin hyödynnettyä paremmin esimerkiksi jakamalla se kahden solmun käyttöön. Myös rajoitettu kolmiointi hyötyisi tästä. Jos vapaan kolmioinnin parannukset toimisivat ja saataisiin toteutettua, oikeastaan ainoa nelipuumenetelmän selvä etu olisi sen yksinkertaisuus.

7 Yhteenveto

Tässä tutkielmassa taustoitettiin korkeuskarttoihin perustuvien kolmiulotteisten maastojen ja planeettojen reaaliaikaista renderöintiä sekä esiteltiin niihin suunniteltuja menetelmiä ja algoritmeja. Myös maaston korkeuskarttojen proseduraaliseen generointiin soveltuvia, kohinafunktioihin perustuvia algoritmeja kuvailtiin, sekä esiteltiin planeettojen suuresta mittakaavasta koituvia teknisiä haasteita ja näihin joitain ratkaisuja. Lisäksi osana tutkimusta kehitettiin kolme hieman toisistaan eroavaa planeettojen renderöintiin tarkoitettua menetelmää, jotka nimettiin alustavan geometrian luontitavan mukaan nelipuumenetelmäksi, vapaaksi kolmioinniksi ja rajoitetuksi kolmioinniksi.

Nelipuumenetelmässä planeetan pohjana on kuutio, jonka jokainen tahko toimii oman nelipuunsa juurisolmuna. Nelipuita tihennetään kameran sijaintiin perustuen sopivan yksityiskohtaisuuden tason saavuttamiseksi. Lopulta nelipuiden lehtisolmut piirretään tiheämmillä kolmioverkoilla. Maaston muodot generoidaan erilaisia Perlin-kohinaan perustuvia summakohinoita yhdistelemällä.

Vapaa kolmiointi on monella tapaa samankaltaisesti kuin nelipuumenetelmä. Siinä myös tihennetään puuhierarkioita sopivan yksityiskohtaisen maaston aikaansaamiseksi. Puiden solmut ovat neliskulmaisten sijaan kolmioita. Planeetan pohjalle soveltuu siis mikä tahansa monitahokas, jonka tahkot on jaettu kolmioiksi. Toteutuksessa planeetan pohjalla käytettiin Kelvinin rakennetta.

Rajoitettu kolmiointi eroaa vain vähän vapaasta kolmioinnista. Puuhierarkioita tihennettäessä pidetään naapurisolmut enimmillään yhden tihennystason päässä toisistaan. Ideana oli piirtää lehtisolmut käyttäen sellaisia kolmioverkkoja, että eri tihennystason solmut yhdistyisivät saumattomasti. Solmut eivät lopulta kuitenkaan yhdistyneet täysin saumoitta.

Menetelmiä verrattiin toisiinsa. Nelipuumenetelmä havaittiin yksinkertaisimmaksi toteuttaa. Rajoitettu kolmiointi oli puolestaan monimutkaisin. Kolmiointiin perustuvien menetelmien etuna on joustavuus planeetan pohjalla olevan monitahokkaan valinnassa. Menetelmien suorituskyvyn ja muistintarpeen mittauksissa nelipuumenetelmä oli tehokkain.

Lähteet

Archer, Travis. 2011. *Procedurally Generating Terrain.* Tekninen raportti. Morningside College, Iowa.

Berg, Mark de, ja Katrin Dobrindt. 1995. "On Levels of Detail in Terrains". Teoksessa *Proceedings of the Eleventh Annual Symposium on Computational Geometry,* 426–427. SCG '95. Vancouver, British Columbia, Canada: ACM. doi:10.1145/220279.220334.

Brodersen, Anders. 2005. "Real-time Visualization of Large Textured Terrains". Teoksessa *Proceedings of the 3rd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia,* 439–442. GRAPHITE '05. Dunedin, New Zealand: ACM. doi:10.1145/1101389.1101477.

Catmull, Edwin. 1974. *A Subdivision Algorithm for Computer Display of Curved Surfaces.* Tekninen raportti. Computer Science Department, University of Utah, Salt Lake City.

Cignoni, P., F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio ja R. Scopigno. 2003. "Planet-sized batched dynamic adaptive meshes (P-BDAM)". Teoksessa *IEEE Visualization, 2003. VIS 2003.* 147–154. doi:10.1109/VISUAL.2003.1250366.

Clasen, Malte, ja Hans-Christian Hege. 2006. "Terrain Rendering Using Spherical Clipmaps". Teoksessa *Proceedings of the Eighth Joint Eurographics / IEEE VGTC Conference on Visualization,* 91–98. EUROVIS'06. Lisbon, Portugal: Eurographics Association. doi:10.2312/VisSym/EuroVis06/091-098.

Cohen-Or, D., E. Rich, U. Lerner ja V. Shenkar. 1996. "A real-time photo-realistic visual flythrough". *IEEE Transactions on Visualization and Computer Graphics* 2 (3): 255–265. doi:10.1109/2945.537308.

Dick, Christian, Jens H Krüger ja Rüdiger Westermann. 2009. "GPU Ray-Casting for Scalable Terrain Rendering." Teoksessa *Eurographics (Areas Papers),* 43–50. Citeseer.

Duchaineau, M., M. Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich ja M. B. Mineev-Weinstein. 1997. "ROAMing terrain: Real-time Optimally Adapting Meshes". Teoksessa *Visualization '97., Proceedings,* 81–88. doi:10.1109/VISUAL.1997.663860.

Foley, James D., Andries van Dam, Steven K. Feiner ja John F. Hughes. 1996. *Computer graphics: principles and practice.* Second Edition in C. Addison-Wesley Publishing Company.

Goldberg, David. 1991. "What Every Computer Scientist Should Know About Floating-point Arithmetic". *ACM Comput. Surv.* (New York, NY, USA) 23 (1): 5–48. doi:10.1145/103162.103163.

Kemen, Brano. 2012. *Maximizing Depth Buffer Range and Precision.* https://outerra.blogspot.com/2012/11/maximizing-depth-buffer-range-and.html.

Kooima, Robert. 2008. "Planetary-scale Terrain Composition". Tohtorinväitöskirja, Graduate College of the University of Illinois.

Lindstrom, Peter, David Koller, William Ribarsky, Larry F. Hodges, Nick Faust ja Gregory A. Turner. 1996. "Real-time, Continuous Level of Detail Rendering of Height Fields". Teoksessa *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques,* 109–118. SIGGRAPH '96. New York, NY, USA: ACM. doi:10.1145/237170.237217.

Losasso, Frank, ja Hugues Hoppe. 2004. "Geometry Clipmaps: Terrain Rendering Using Nested Regular Grids". Teoksessa *ACM SIGGRAPH 2004 Papers,* 769–776. SIGGRAPH '04. Los Angeles, California: ACM. doi:10.1145/1186562.1015799.

Mandelbrot, Benoit B. 1982. *The fractal geometry of nature.* Nide 1. WH freeman New York.

Mantler, Stephan, ja Stefan Jeschke. 2006. "Interactive Landscape Visualization Using GPU Ray Casting". Teoksessa *Proceedings of the 4th International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia,* 117–126. GRAPHITE '06. Kuala Lumpur, Malaysia: ACM. doi:10.1145/1174429.1174448.

Miller, Gavin S P. 1986. "The Definition and Rendering of Terrain Maps". Teoksessa *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques,* 39–48. SIGGRAPH '86. New York, NY, USA: ACM. doi:10.1145/15922.15890.

Musgrave, F. K., C. E. Kolb ja R. S. Mace. 1989. “The Synthesis and Rendering of Eroded Fractal Terrains”. Teoksessa *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques,* 41–50. SIGGRAPH ’89. New York, NY, USA: ACM. doi:10.1145/74333.74337.

Newell, M. E., R. G. Newell ja T. L. Sancha. 1972. “A Solution to the Hidden Surface Problem”. Teoksessa *Proceedings of the ACM Annual Conference - Volume 1,* 443–450. ACM ’72. Boston, Massachusetts, USA: ACM. doi:10.1145/800193.569954.

NVIDIA. 2017. *NVIDIA Tesla P100.* Whitepaper v1.2. Haettu 6.6.2017.

Perlin, Ken. 1985. “An Image Synthesizer”. *SIGGRAPH Comput. Graph.* (New York, NY, USA) 19 (3): 287–296. doi:10.1145/325165.325247.

———. 2002. “Improving Noise”. Teoksessa *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques,* 681–682. SIGGRAPH ’02. San Antonio, Texas: ACM. doi:10.1145/566570.566636.

Porwal, Sudhir. 2013. “Quad Tree-based Level-of-details Representation of Digital Globe”. *Defence Science Journal* (METCALFE HOUSE, DELHI 110054, INDIA) 63 (1, SI): 89–92.

Qu, H., F. Qiu, N. Zhang, A. Kaufman ja M. Wan. 2003. “Ray tracing height fields”. Teoksessa *Proceedings Computer Graphics International 2003,* 202–207. doi:10.1109/CGI.2003.1214467.

Sellers, Graham, Juraj Obert, Patrick Cozzi, Kevin Ring, Emil Persson, Joel de Vahl ja J. M. P. van Waveren. 2013. “Rendering Massive Virtual Worlds”. Teoksessa *ACM SIGGRAPH 2013 Courses,* 23:1–23:88. SIGGRAPH ’13. Anaheim, California: ACM. doi:10.1145/2504435.2504458.

Shoemake, Ken. 1985. “Animating Rotation with Quaternion Curves”. Teoksessa *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques,* 245–254. SIGGRAPH ’85. New York, NY, USA: ACM. doi:10.1145/325334.325242.

Ulrich, Thatcher. 2002. “Rendering massive terrains using chunked level of detail control”. *SIGGRAPH Course Notes* 3 (5).

Liitteet

A Kuvia planeetasta