Tampereen ammattikorkeakoulu

Interaktiivisen maaston luominen 3D-moottorissa

Santeri Sivula

OPINNÄYTETYÖ
Joulukuu 2020

Tietojenkäsittely
Game Production

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Game Production

SIVULA, SANTERI:
Interaktiivisen maaston luominen 3D-moottorissa

Opinnäytetyö 21 sivua, joista liitteitä 0 sivua
Joulukuu 2020

Tämän opinnäytetyön toimeksiantajana toimi Varaani Works Oy. Opinnäyteprojektin tavoitteena oli kartoittaa erilaisia mahdollisuuksia luoda maastoa 3D-moottoreissa. Pyrkimyksenä oli luoda toteutus, jota käyttäjä pystyy muokkaamaan ohjelmaa käyttäessään.

Työssä tutkittiin nykypäiväisten 3D-moottorien sisäänrakennettuja maastonluonnin mahdollisuuksia, ja todettiin, että ne eivät olleet riittävän monipuolisia toteuttamaan interaktiivista maastoa ilman suuria ponnisteluja.

Tuloksena luotiin interaktiivista maastoa Unity-nimiseen 3D-moottoriin, joka voidaan implementoida projekteihin, jotka sitä tarvitsevat. Toteutuksessa on käytetty Marching Cubes -nimistä pinnanrakennus algoritmia, rakentamaan maaston pinnan abstraktista datasta, jota voidaan luoda ohjelman käytön aikana.

Asiasanat: interaktiivinen maasto, marching cubes, unity

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Game Production

SIVULA, SANTERI:
Creating Interactive Terrain in a 3D-engine

Bachelor's thesis 21 pages, appendices 0 pages
December 2019

This thesis was commissioned by Varaani Works Ltd. to survey different possibilities to create terrain in a 3D-engine and to create terrain, which the end user can modify while using the program it is in.

In this thesis, implementations of terrain from modern 3D-engines were researched. It was discovered that they were lacking some features required to create interactive terrain without major effort.

As a result of this thesis, interactive terrain was created into Unity 3D-engine, which can be implemented to projects as needed. In the implementation of the terrain, the algorithm Marching Cubes was used to create the surface of the terrain from abstract data.

Key words: interactive terrain, marching cubes, unity

SISÄLLYS

ERITYISSANASTO tai LYHENTEET JA TERMIT (valitse jompikumpi)

3D	Kolmiulotteinen
2D	Kaksi ulotteinen
Verteksi	kolmiulotteisen avaruuden piste
Tekstuuri	Kaksi ulotteinen kuva, joka piirretään kolmiulotteisen mallin päälle
UV	Verteksille annettavat koordinaatit tekstuurista, jotka tarvitaan tekstuurin oikein piirtämisen vuoksi
Bitti	Binäärinen numero, 1 tai 0
Tavu	8 Bittiä

Aloita kirjoittaminen yllä olevien esimerkkien paikalta. Poista tämä sivu, mikäli opinnäytetyössäsi ei tarvita erityissanastoa, lyhenteitä tai termejä.

1 JOHDANTO

Tämän opinnäytetyön tarkoitus oli toteuttaa toimiva interaktiivinen maasto 3D peli moottorissa. Käytetty moottori opinnäytetyössä on Unity sen yleisyyden sekä tuttuuden vuoksi. Toimeksiantaja Varaani Works Oy halusi kartoittaa erilaisia menetelmiä maastonluontiin 3D ympäristössä ja selvittää kuinka toteuttaa interaktiivista maastoa.

Opinnäytetyössä on käytetty Marching Cubes nimistä pinnan rakennus algoritmia, joka on julkistettu vuonna 1987 lääketieteelliseen käyttöön sen suhteellisen yksinkertaisuuden vuoksi.

Työn pääpainona on toteutuksen esittely. Tavoitteena on esitellä kuinka toteuttaa interaktiivista maastoa pelimoottorissa ja sen lisäksi tutkia muita mahdollisia käyttötarkoituksia algoritmille 3D peli moottorissa.

Opinnäytetyössä esitellään ensin lyhyesti erilaisia tapoja toteuttaa maastoa, sen jälkeen perehtyä pinnanrakentamiseen. Seuraavaksi käsitellään toteutuksen rakenne. Lopuksi pohditaan parannuksia, jatkokehitysmahdollisuuksia ja muita käyttötarkoituksia.

2 MAASTONLUONTI

2.1 Staattinen maasto

Viihde- ja teollisuus aloilla on tarvittu virtuaalisesti luotua maastoa jo kauan. Esimerkiksi teollisuus alalla simulaattoreista koulutusvideoihin, ja viihdealalla elokuvissa taustalla olevasta maastosta, videopeleissä oleviin massiivisiin tutkittaviin maastoihin. Armeija käytössäkin maastonluonnilla on iso osa sotasimulaattoreissa uusien ja realistisien skenaarioiden luomiseksi. (Guo ym. 2019, 2)

Maastoa pystyy luomaan monilla eri tavoilla 3D- Moottoreihin, eri tavoilla on omat hyötynsä ja haittansa liittyen käytettävyyteen, ylläpidettävyyteen ja lopputuloksen ilmeeseen. Kun valitaan eri tavoista, pitää huomioida, kuinka maaston pitää käyttäytyä muiden projektissa olevien elementtien kanssa ja tunnistaa mahdolliset riskit elementtien välisistä kanssakäymisistä. (Smelik ym. 2010)

Mallinnus ohjelmistoilla esim. blender, pystyy tuottamaan maastoa ja tuomaan sen yleisimpiin 3D moottoreihin suhteellisen helposti ja vaivattomasti, sillä 3D moottorit tukevat ulkopuolisten mallien sisään tuonnin vakio ominaisuutena. Mutta jos maastoa luodaan toisessa ohjelmassa, ja sitä halutaan päivittää, pitää päivittäminen tehdä käytetyssä ohjelmassa ja lisätä malli uudestaan moottoriin. (Unity User Manual)

Nykypäiväisissä 3D moottoreissa on sisäänrakennettuja maastonluonti kirjastoja, jotka pääsääntöisesti perustuvat korkeuskarttoihin. Korkeuskartat ovat kaksiulotteisia kuvia, missä yksittäisen pikselin väriarvo jollain skaalalla tarkoittaa tiettyä korkeutta. Yleisimmät korkeuskartat ovat musta valkoisia, missä musta tarkoittaa nolla korkeutta ja valkoinen on erikseen määritelty maksimi korkeus. (Unity User Manual)

Korkeuskartat eivät itsestään tue luolastoja tai maaston päälle tulevia ulokkeita. Mutta on olemassa erilaisia tekniikoita, joilla pystytään kiertämään tällaiset rajoitteet. Esimerkiksi luolan voi toteuttaa, tekemällä osasta maastoa näkymätöntä ja laittamalla mallin sen taakse. (Alicea A. 2020.)

2.2 Interaktiivinen maasto

Tässä työssä interaktiivisella maastolla tarkoitetaan maastoa, jota loppukäyttäjä pystyy jotenkin muokkaamaan, esimerkiksi, simulaatiossa siirtämään hiekkaa kaivinkoneella tai kaivamaan luolastoja maan alle räjähteillä. Vaikka maaston pinta muuttuu koko ajan pitäisi sen olla sulavan näköistä ja olla vaikuttamatta projektin suorituskykyyn liikaa, ettei sovellus jäätyisisi.

Interaktiivisen maaston luomiseksi tarvitaan maasto dataa ja pinnanrakennus algoritmi. Se miten maastodata on tallennettu muistiin, ei ole mitään merkitystä, kunhan datasta pystytään katsomaan yksittäisiä pisteitä, jotka kertovat, onko kyseisessä pisteessä mitään. (GPU Gems 3. Ch 1)

Pinnanrakennus algoritmeja on monia, ja niiden tuottamien pintojen laatu voi vaihdella paljon. (MacDonald T.) Kun lähtee suunnittelemaan, mitä pinnanrakennus algoritmia tulee käyttämään, kannattaa ensin tutkia, minkä tasoista pintaa halutaan.

3 PINNANRAKENTAMINEN

3.1 Pinnanrakentamisen periaatteet 3D moottorissa

Pinnanrakentamisen periaatteena on luoda visuaalinen malli abstraktista datasta esim. kolmiulotteisesta skaalari kentästä. (Bourke, 1994) Lopputulos 3D moottorissa silloin koostuu verteksi pisteistä ja kolmiotaulukosta. Pelkistä Verteksi pisteistä ja kolmiotauluista koottu malli ei vielä ole valmis, sillä pitää jokaiselle verteksi pisteelle kertoa, mikä on kyseisen pisteen pinnan suunta eli verteksin normaali. Verteksien normaalien avulla pystytään laskemaan fysiikka pohjaisten törmäyksien suuntia ja kuinka valon tulisi kimmota pinnasta, kun sitä katsotaan. (Schrute. 2019)

Verteksi normaalien jälkeen mallin vertekseiltä puuttuu vielä UV kartoitukseen vaadittavat koordinaatit mitkä kertovat mistä kohtaa ennalta määritettyä kaksiulotteista kuvaa tulisi piirtää mallin päälle, että saataisiin esimerkiksi mallin pinta näyttämään hiekalta tai ruohikolta. (Schrute. 2019)

Vaikka pinnanrakentamista varten on kehitetty monta eri algoritmia, tulemme tässä työssä vain keskittymään niistä yhteen, mitä on käytetty toteutuksessakin.

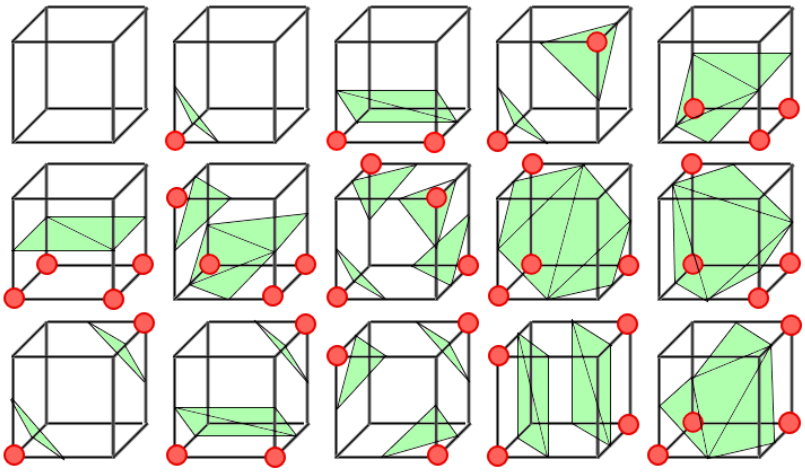
3.2 Marching Cubes Algoritmi

Marching Cubes algoritmi perustuu hakutauluihin ja tämän takia algoritmi on nopea sekä selkeä. Algoritmi on saanut nimensä sen toiminta mallistaan, sillä algoritmin ajaminen muistuttaa “marssivia” kuutioita, jotka rakentavat palan pintaa, ja tämän jälkeen liikkuu mittansa verran eteenpäin ja rakentaa uuden palan. Kun data on käyty kokonaan lävitse, yhdistetään tuotetut palaset kokonaiseksi malliksi, ks. kuva 2. (Lorensen W E & Cline H E. 1987.)

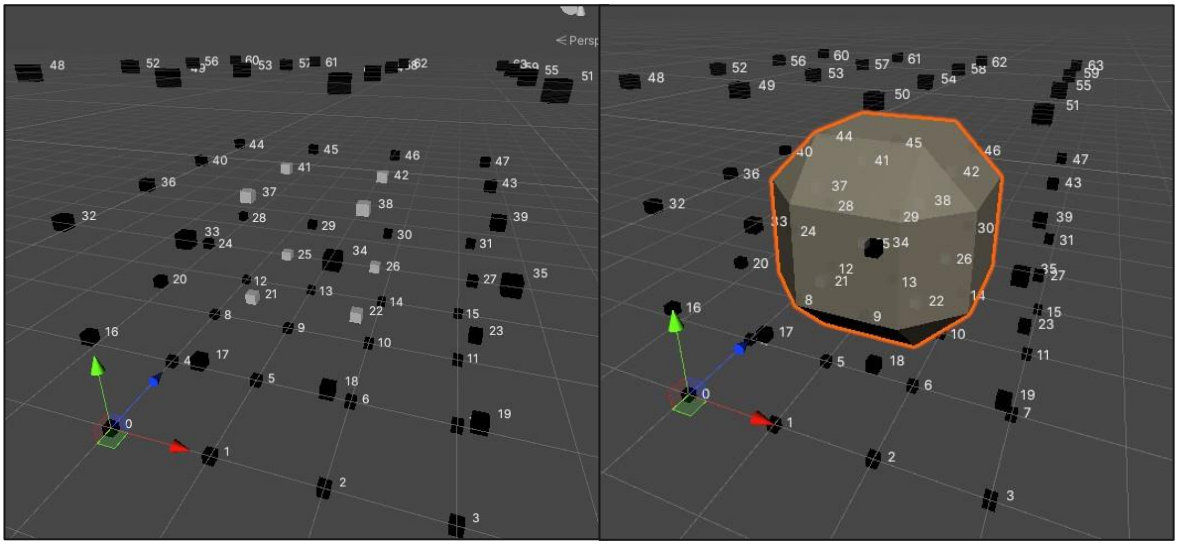
“Kuutiot” rakentavat paloja tarkistamalla jokaisen kulmansa datan tilan ja sen perusteella haetaan hakutaulusta verteksien sijainnit ja kolmiotaulut. Koska kuu-

tioissa on vain kahdeksan kulmaa, mahdollisten yhdistelmien määrä on 2^8. Tämän vuoksi voimme hyödyntää yhtä tietoteknistä tavua seuraamaan kuution tilaa. (Lorensen W E & Cline H E. 1987.)

Mahdollisista yhdistelmistä voimme tunnistaa viisitoista erilaista uniikkia konfiguraatiota, ks. kuva 1, jotka toistuvat moneen kertaan. Esimerkiksi jos merkitsevää dataa löytyy vain yhdestä kulmasta, luodaan yksi kolmio nurkkaan. Tämä tapahtuu kaikista yhdistelmästä kahdeksan kertaa. Kaikista mahdollisista konfiguraatioista voimme jättää kaksi täysin huomiotta, konfiguraation missä ei ole yhtään merkittävää data pistettä, ja konfiguraation missä jokaisessa pisteessä on merkittävää dataa. Sillä voidaan sanoa, että ne ovat kokonaan “pinnan” ulkopuolella tai sisällä. (Lorensen W E & Cline H E. 1987.)



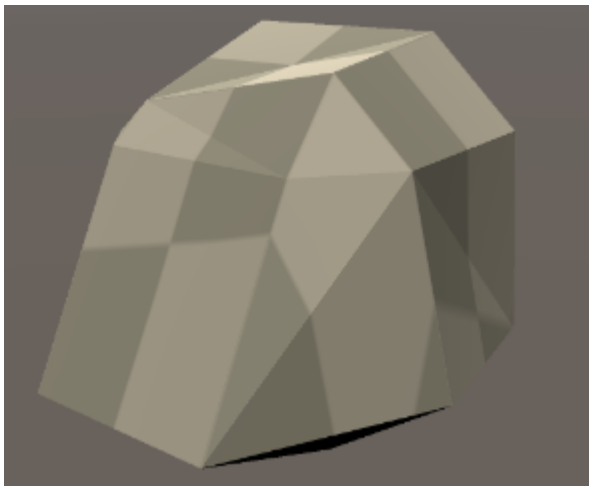
Kuva 1. Marching Cubes algoritmin 15 eri uniikkia konfiguraatiota



Kuva 2. Kuutio ilman kolmioita (vas.), “Kuutio” algoritmin ajon jälkeen(oik.)

Kun erilaisia konfiguraatioita tutkii, tulee myös huomattua, että uniikkeja konfiguraatioita yhdistelemällä, lopputuloksessa jokainen kulma on 45°. Tämä pystytään välttämään antamalla jokaiselle “kuution” nurkalle painoarvoja, sen sijaan että nurkassa on tai ei ole dataa. Nurkkien painoarvojen avulla pystytään laskemaan kuinka paljon yksi nurkka vaikuttaa laskentaan suhteessa muihin nurkkiin. (Bourke P. 1994.)

Kun painoarvoja käytetään pitää luoda määritelmä siitä, mikä lasketaan merkittäväksi. Esimerkki, Jos nurkan arvo on yli 0.5 lasketaan se merkittäväksi, ja jos arvo on yli 1, otetaan se huomioon yhtä paljon kuin 1. Kun painot on määritelty tällä tavalla, pystymme kiertämään 45° rajoituksen, kuten kuvassa 3, voidaan huomata.



Kuva 3. “Kuutio” nurkka painojen ja pinta tekstuurin kanssa

4 MAASTODATA

4.1 Ennalta määritetty data

Jotta voitaisiin rakentaa maastoa pinnanrakennus algoritmilla, tulee algoritmille toimittaa dataa mistä pinta rakennetaan. Data voi olla skannattua, mallinnettua tai satunnaisesti generoitua. Datan pitää olla algoritmille ystävällisessä muodossa, tässä toteutuksessa data pisteiden välit pitää olla saman mittaisia, ja arvot saavat vaihdella nollan ja yhden välillä, datan rakenteen siis pitää muistuttaa pistepilveä, jossa pisteet ovat tasavälein.

Kolme ulotteisesta mallista voidaan luoda tällaista dataa, esimerkiksi mittaamalla tasaisin välein, kuinka paljon massaa kyseisessä kohdassa on. Laserkeilattu maasto ei välttämättä suoraan toimisi, vaikka laserkeilauksesta tuotettu data on pitkälti pistepilvi, sitä jouduttaisiin todennäköisesti muokkaamaan käyttämään painoarvoja ja pisteiden olemaan tasavälein.

4.2 Näennäissatunnaisesti luotu data

Näennäissatunnaisesti luodulla datalla tarkoitetaan dataa, jolle annetaan arvoja, joiden perusteella luodaan dataa. Tämä myös tarkoittaa sitä, että jos annetaan samat arvot kahteen kertaan, pitää tuloksen olla täysin sama. Tällaista dataa hyödynnetään pääsääntöisesti peli maaston luomiseen, minkä pitää olla loputon. (Rose T J, Bakaoukas A G. 2016.) Hyvänä esimerkkinä tähän toimii peli Minecraft, sillä pelaaja voi mennä jokaiseen ilman suuntaan pelaajan näkökulmasta loputtomasti ja uutta maastoa luodaan pelaajan eteen.

Vaikka on puhe maasto datasta, ei sen tarvitse kuvata oikean elämän maastoa. Data voi esimerkiksi kuvata luolastojakin, tätä voidaan hyödyntää esimerkiksi luolasukellus simulaatioissa täyttämällä luodun maaston vedellä.

5 INTERAKTIIVINEN MAASTO UNITYSSÄ

5.1 Unity lyhyesti

Unity on Unity Technologiesin valmistama 3D pelimoottori ja siihen sisällöntuottamiseen luotu editori. Unity oli aluksi pelkästään 3D pelejä varten luotu, silloin kun kolmiulotteisten ohjelmien tarve nousi muillakin kuin peli alalla, ovat he panostaneet muuhunkin kuin pelkkään peli puoleen.

Unityllä pystyy nykyään luomaan animoituja elokuvia, pelejä, simulaatioita ja muita kolmesta ulottuvuudesta hyötyviä ohjelmia. Vaikka kolmiulotteisuus on vahvasti esillä Unityssä, pystytään sillä luomaan tarvittaessa myös kahden ulottuvuuden ohjelmia kolmen sijaan.

Unity on suosittu harrastelijoiden, että ammatillisten keskuudessa, jotka tuottavat pelejä tai muuta kolmesta ulottuvuudesta hyötyvää ohjelmaa, monipuolisuutensa ja selkeän rakenteensa vuoksi.

5.2 Unityn maastonluonnin kirjastot

Unityssä on valmiina jo maastonluontia tukevia kirjastoja ja työkaluja, mutta ne toimivat korkeuskarttojen varassa, mikä tarkoittaa sitä, että niillä luotujen maastojen interaktiivisuus on vaikea toteuttaa. Esimerkiksi, jos maastoon halutaan luoda reikä, luodaan valitun kohdan pinnasta läpinäkyvä ja pitää sen taakse itse luoda erillisistä malleista, tai muulla tavalla, luolan sisusta.

Muuten kirjastot ja työkalut mitä Unitystä jo valmiiksi löytyy, sopeutuu hyvin staattisen maaston tekemiseen, sillä korkeuskarttojen lisäksi, maastoon pystyy lisäämään värejä ja tekstuureja, jotka sekoittuvat hyvin keskenään. Ja asettamaan malleja mitä on tuotu projektiin, esimerkiksi kiviä ja puita.

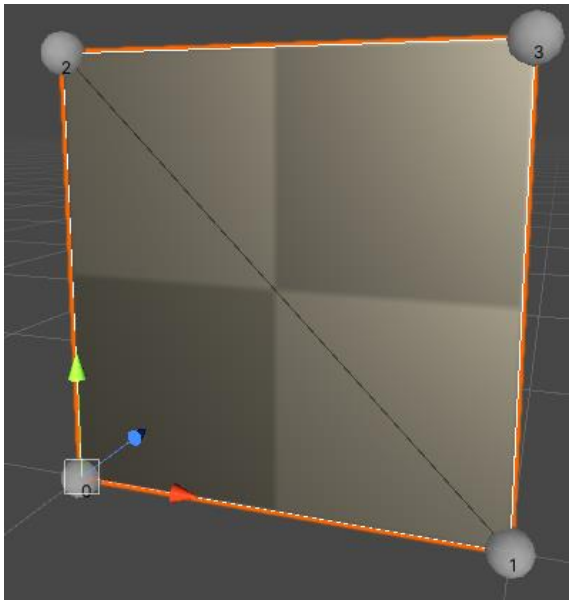
5.3 Unity ja pinnanrakennus

Unityssä ei ole sisäänrakennettua toiminnallisuutta pinnanrakentamisella, joten toiminnallisuus pitää tuoda Unityyn ulkopuolelta. Esimerkiksi, kirjoittamalla sitä tukeva kirjasto itse.

Kun pinnanrakentamista lähtee toteuttamaan, pitää ensin tietää kuinka kolmiulotteiset mallit toimivat Unityssä. Unityssä käytetään pääsääntöisesti pelkkiä pinta malleja, eli kaikki mallit ovat onttoja sisältä ja vain mallin pintaa käsitellään. Tämä vapauttaa käytettävissä olevaa laskentatehoa muihin tehtäviin, esimerkiksi, pinnan rakentamiseen.

Pinta mallit koostuvat pääsääntöisesti vertekseistä ja kolmiotaulusta. Yksittäinen verteksi sisältää tiedon sijainnista kolmiulotteisessa tilassa ja kolmiotaulu on yksiulotteinen taulukko, joka on jaollinen kolmella ja sisältää vain verteksien järjestys numeroita. Kolme peräkkäistä solua kolmiotaulussa kertoo Unitylle, mihin kohtaan yhden kolmion nurkat tulevat. Kun Unity piirtää malleja ruudulle, pitää ottaa huomioon, että Kolmioilla on vain yksi näkyvä pinta, koska Unityssä käytetään “Back-face Culling”: ia, eli piirretään vain kolmioiden etupuoli, jotka määritetään kolmiotaulussa olevien numeroiden järjestyksellä. Järjestyksen pitää näyttää siltä, että järjestys pyörisi myötäpäivään. Jos järjestys on väärä, kolmio näkyy vain toiselta puolelta.

Kuvassa 4, on “Quad”, jossa on neljä verteksi pistettä ja kaksi kolmiota. Kolmiot voivat jakaa verteksi pisteitä, mutta Unity suosii malleja, jossa verteksi pisteitä ei jaeta, koska silloin jokaiselle verteksille voidaan määrittää omat UV arvot tekstuurien piirtämistä varten.



Kuva 4. Kolmiulotteinen neliö, joka koostuu kahdesta kolmiosta, kolmiotaulun sisältö [0, 2, 1] + [2, 3, 1]. Mukana myös UV koordinaatit ja Normaalit

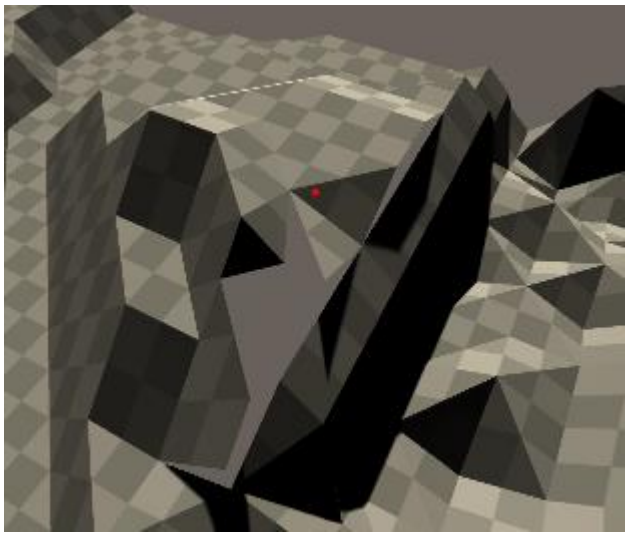
Kolmiulotteisiin malleihin tarvitaan myös enemmän dataa, että saataisiin ne näkymään paremmin, tarvitsevat ne Normaalit ja UV koordinaatit. Normaalit ovat vertekseihin liitettyjä suunta vektoreita, jotka osoittavat pinnasta ulospäin. Normaaleja käytetään esimerkiksi valon vaikutuksen laskemiseen pintaan. UV koordinaatit ovat jokaiselle verteksille määrätty suhteellinen kaksiulotteinen sijainti kuvasta mitä pitäisi piirtää mallin päälle. Unityssä kun luo malleja lennosta, voidaan normaalien laskeminen jättää Unityn tehtäväksi. UV koordinaattien laskemin voi taas tapahtua monella eri tapaa, kaikista helpoin on määrä UV koordinaatiksi pisteen sijainniksi verteksin x ja z sijainti. Mutta se tarkoittaa sitä, että tekstuuri, jota piirretään päälle, piirtyy oikein vain ja ainoastaan x ja z skaalalla, jos mallissa tulee mukaan y koordinaatin muutoksia, tulee tekstuuri venymään sellaisen kohdalla.

5.4 Unity ja Marching Cubes

Kun toteuttaa Marching Cubes algoritmia Unityyn, pitää ottaa huomioon muutama suorituskyvyllistä seikkaa. Ensinnäkään jos koko massan määrä on suuri, ei kannata ajaa algoritmia kokomassalle, muuten kuin alustuksen yhteydessä. Tämän jälkeen, kun pinta pitää rakentaa uudestaan, kannattaa rakentaa vain osa muunnetusta pinnasta uudestaan. Yleisin tapa tähän on jakaa kokomassa pienempiin lohkoihin, esimerkiksi jos massan data olisi 100^3 kooltaan, voidaan lohkon kooksi määrittää 10^3. Kun jaetaan data lohkoihin, voidaan suorituskykyä parantaa vielä enemmän, jos lohkoihin ajetaan algoritmia rinnakkain.

Datan lohkottamisen lisäksi, ei kannata algoritmia ajaa jokaisella Unityn päivitys syklillä, muuten projekti jäätyy ja on käyttökelvoton. Joten voimme laittaa jokaiselle lohkolle merkin, jos sitä on muutettu, ja päivitys syklin yhteydessä, päivittää lohkon rakentama pinta, jos merkki löytyy lohkosta.

Kun data jaetaan lohkoihin, helpoin virhe toteuttaa on se, että lohkot eivät tiedä naapureistaan mitään. Tämä pitää ottaa huomioon, kun pinta data luodaan, muuten lohkojen välit voivat “irvistellä”, kuten kuvassa 5, voidaan huomata. Asian voi korjata, määrittelemällä lohkojen rajat siten, että jokainen reuna, jossa on naapuri, jakavat data pisteensä keskenään.

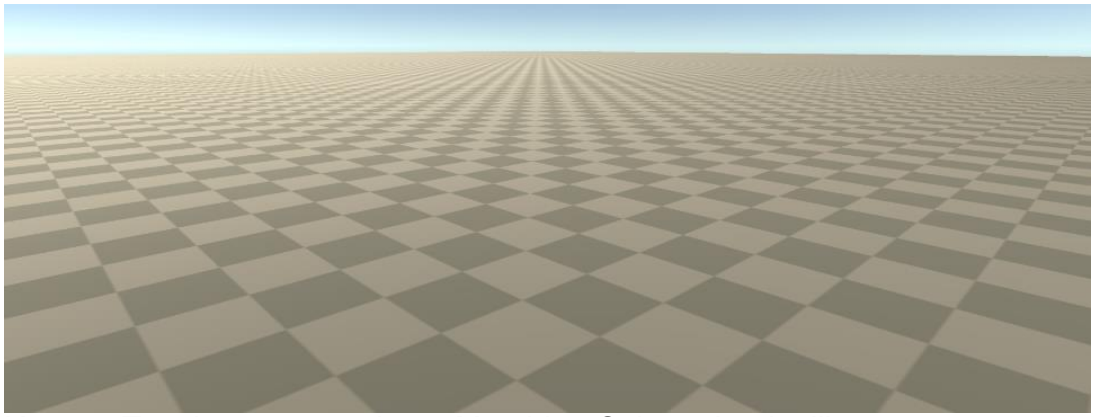


Kuva 5. “irvistävä” lohko

5.5 Muokattavan pintadatan luominen

Kun käytetään Marching Cubes algoritmia, voimme Unityssä luoda pintadatan helposti käsiteltäväksi ja ymmärrettäväksi, luomalla data pisteet tasaisin välein kolmiulotteisessa tilassa, koska se helpottaa datan visualisointia ilman algoritmin ajamista.

Alkuun on hyvä luoda tasainen pinta, ks. kuva 6, joka menee jossain tietyssä korkeudessa, ja lähteä sen päälle muokkaamaan erikeinoin mielenkiintoisimpia ominaisuuksia. Tasaisen pinnan saa aikaiseksi, kun jokaiselle datapisteelle mikä on tietyn rajan alla, antaa maksimi painon.

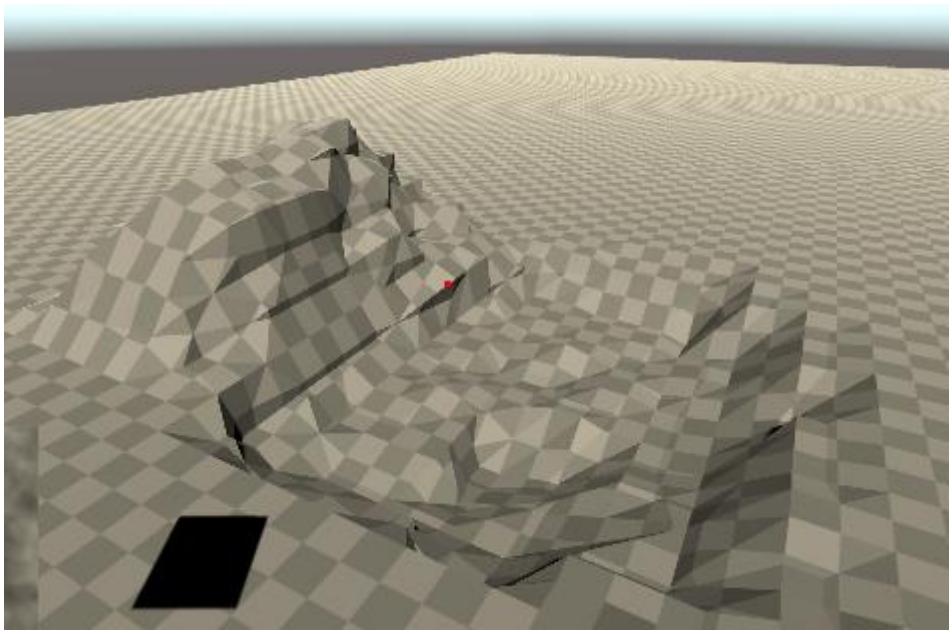


Kuva 6. Tasainen maasto luotu Marching Cubes algoritmilla.

Kun tasainen maasto on saatu luotua, voidaan toteuttaa työkalu, jolla maastoa muokataan. Työkalu voi olla visuaalisesti mitä vain, mutta tässä toteutuksessa on käytetty näkymätöntä laser “pistoolia”, joka nostattaa tai laskee maan pintaa, siitä kohdasta mihinkä sillä on osuttu. Alue johon laser vaikuttaa pitää myös selvittää, sillä jos laserin vaikutusalue on pienempi kuin maastodata pisteiden välinen tila, ei laser pysty muokkaamaan maastodataa luotettavasti.

Työkalun vaikutusalue voi olla muodoltaan mitä vain, mutta helpoimmat muodot mitä voidaan toteuttaa ovat pallot ja kuutiot, sillä niiden laskeminen on helppoa. Työkalulle on myös hyvä antaa “teho” mikä määrittää sen, kuinka paljon tulee se vaikuttamaan maaston painoihin. Jos “teho” on korkeampi tai yhtä suurikuin maasto datapisteen maksimi paino, poistetaan silloin vaikutusalueelta kaikki painot ja pinnanluonnin yhteydessä, näyttää se siltä, että olisi vain poistettu alueen

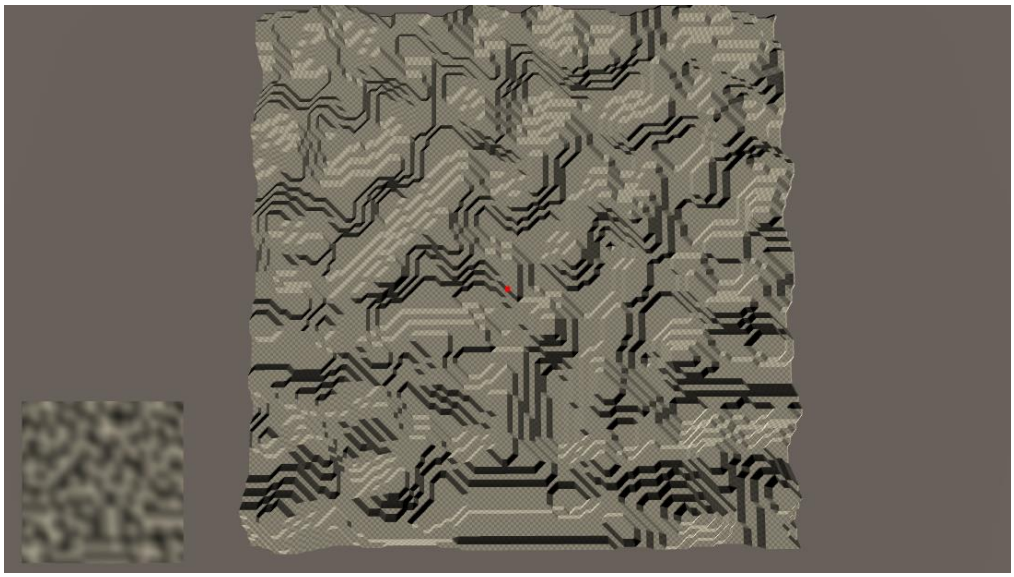
kokoinen pala. Jos työkalun “teho” on kymmenes osa maaston datapisteiden painosta, tapahtuu muutokset maastossa paljon sulavammin, ks. kuva 7.



Kuva 7. Tasaista maastoa, mitä on muokattu maaston kymmenesosa painolla.

5.6 Kohinan lisääminen pintadataan

Kohinalla tarkoitetaan tässä työssä näennäissatunnaisesti luotua kuvaa, jota voidaan verrata korkeuskarttaan. Mikä tarkoittaa sitä, että kuvasta otetaan yksittäisten pikselien väriarvo, ja sen perusteella nostetaan maastoa, ks. kuva 8.



Kuva 8. Kohina kuvan käyttäminen korkeuskarttana

Kohinan luontiin on olemassa monia algoritmia, mutta maaston luonnin yhteydessä tunnetuimmat ovat “Perlin Noise” ja “Simplex Noise”. (Patel A. 2020.) Kummatkin algoritmeista on kehittänyt Ken Perlin, ensin oli “Perlin noise”, mutta se oli hänen mielestään puutteellinen, joten hän kehitti “Simplex noise”:n, joka ei olisi yhtä puutteellinen. (Gustavson S. 2005.) Tässä työssä käytetään “Perlin Noise” kohinaa, sillä se helpommin ymmärrettävissä kuin “Simplex Noise”.

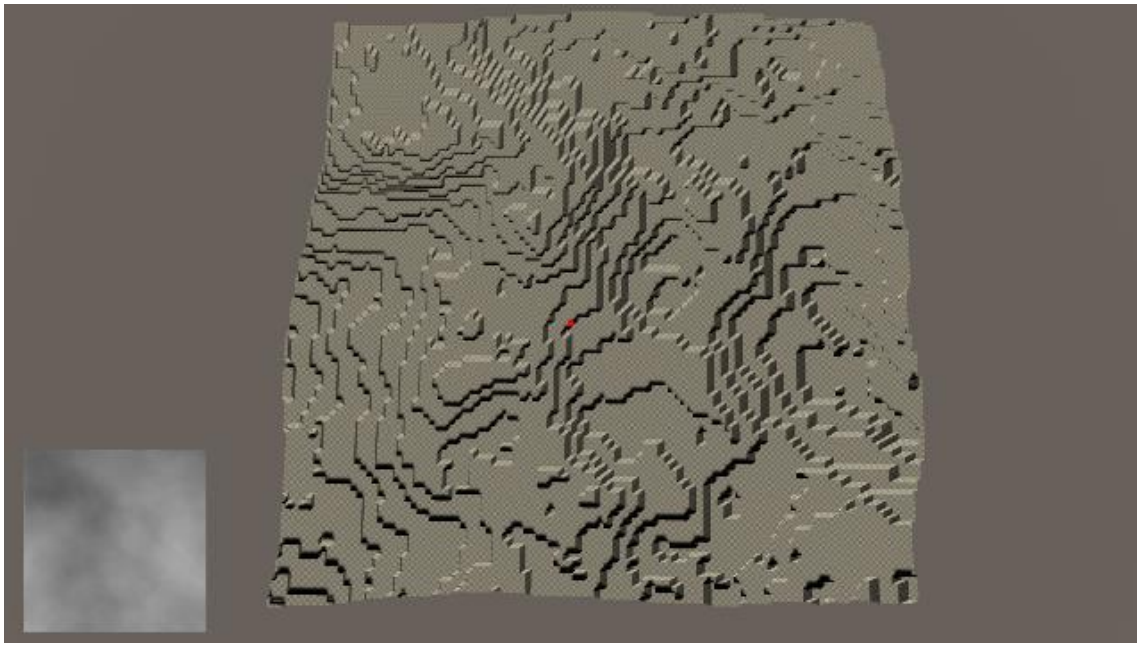
“Perlin Noise” algoritmin tavoitteena on antaa näennäissatunnainen luku, jokaisesta syötteestä, mitä sille annetaan, mikä varmistaa sen, että saadaan samanlaisia tuloksia algoritmilta, jos niitä tarvitaan.

Algoritmin toteuttamiseen tarvitaan algoritmin lisäksi jonkin tasoinen tiivistefunktio. Yksinkertaisin funktio, mitä käytetään tässäkin työssä, on yksinkertainen hajautustaulu. Hajautustaulu on simppelimmillään yksi ulotteinen taulu kokonaislukuja, jotka vastaavat taulun solujen järjestys numeroita, mutta sekoitettuna. Taulun käyttäminen tapahtuu siten, kun numero halutaan muuttaa taulua käyttäen, tarkistetaan vain, mikä numero täsmää tätä numeroa, esim. 1 => 8. Hajautustaulua luodessa, kannattaa ottaa huomioon, ettei sille syötetä arvoja, joita sieltä ei löydy.

Kun hajautustaulu on olemassa, voimme syöttää sen lävitse esimerkiksi koordinaatteja, joista halutaan kohina arvo. Mutta koordinaatit menevät jossain vaiheessa yli tai ali hajautustaulusta, joten koordinaatteja pitää vielä vähän käsitellä, ennen kuin voimme laittaa koordinaatit hajautustaulun lävitse. Yksi tapa, miten varmistaa, että pysymme hajautustaulun sisällä, on muuttaa koordinaatit tasaluvuiksi, ja tämän jälkeen, jos luku on yli taulukon koon, miinustetaan siitä taulukon pituus, ja jos liian pieni, lisätään taulukon pituus. Tämä vaikuttaa myös dataan, miltä se tulee näyttämään, kun sitä visualisoidaan, joten hajautustaulun tarvitsee olla tarpeeksi suuri, ettei toistuvaa kuviota huomattaisi.

Jotta saataisiin kohinasta parempi laatuista maastonluontia varten, voimme “pehmentää” kohina arvoja vertaamalla niitä niiden “vieressä” oleviin arvoihin. Eli ennemmin kuin algoritmi palauttaa arvoa, katsoo se viereisen arvon ja ottavat niiden välisen keskiarvon. Kun pehmennetty arvo on saatu, voimme alkuperäistä arvoa muokata vähän, ja laittaa se uudestaan algoritmiin, ja tuottaa sen pohjalta uusi

kohina arvo. Sitten voimme lisätä kaksi tuotettua arvoa keskenään, täten luoden enemmän muuntelua itse kohina arvoon, ks. Kuva 9.



Kuva 9. Maasto, joka on luotu monen pehmennetyn kohinan yhdistelmästä

6 POHDINTA

Opinnäytetyötä tehdessäni perehdyin Unityn mallien toimintaan paremmin, kuin mitä muuten olisin. Samaten erilaisia maastonluonnin ratkaisuja, kirjastoja ja palveluita tuli tutkittua paljon. Vaikka algoritmit, joita on hyödynnetty ovat vanhempia kuin minä, löytyy niiltä vielä käyttöarvoa nykypäivänäkin.

Toteutuksessa kun on käytetty vanhempia, helpommin ymmärrettäviä algoritmeja, koen toteutuksen tekemisen toimineen itselleni hyvänä motivaattorina lähteä tutkimaan nykyaikaisempia ratkaisuja.

Tuotosta voitaisiin vielä jatko kehittää, sillä kohinasta luodulla korkeuskartalla, korkeuserot mitä maastoon tulee, näyttää vielä epäluonnolliselta. Muuten toimeksiantaja oli tyytyväinen tuotettuihin tuloksiin ja rupesi jo miettimään mahdollisia käyttötarkoituksia enemmän.

LÄHTEET

Guo H, Goodchild M F. & Alessandro A. 2019. Manual of Digital Earth. Singapore: Springer.

Smelik R M, Tutenel T, Kraker K J & Bidarra R. 2010. Declarative Terrain Modeling for Military Training Games. Luettu 10.12.2020. https://doi.org/10.1155/2010/360458

Unity User Manual. Luettu 10.12.2020, https://docs.unity3d.com/Manual/index.html

Alicea A. 2020. Digging into Terrain Paint Holes in Unity 2019.3. Luettu 10.12.2020. https://blogs.unity3d.com/2020/01/31/digging-into-terrain-paint-holes-in-unity-2019-3/

Gerrit G. 2009. Interactive voxel terrain design using procedural techniques. Luettu 10.12.2020. http://hdl.handle.net/10019.1/2631

Nvidia. GPU Gems 3. Luettu 10.12.2020. https://developer.nvidia.com/gpugems/gpugems3/contributors

MacDonald T. Isosurface Extraction. Luettu 10.12.2020. https://swiftcoder.wordpress.com/planets/isosurface-extraction/

Bourke P. 1994. Polygonising a scalar field. Luettu 10.12.2020. http://paulbourke.net/geometry/polygonise/

Schrute A. 2019. Understanding Mesh anatomy in Unity. Luettu 10.12.2020. https://medium.com/shader-coding-in-unity-from-a-to-z/understanding-mesh-anatomy-in-unity-2507cb1ae011

Lorensen W E & Cline H E. 1987. Marching Cubes: a high resolution 3D surface construction algorithm. Luettu 10.12.2020. https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.545.613

Rose T J, Bakaoukas A G. 2016. Algorithms and Approaches for Procedural Terrain Generation - A Brief Review of Current Techniques. Luettu 10.12. 2020. https://doi.org/10.1109/VS-GAMES.2016.7590336

Patel A. 2020. Making maps with noise functions. Luettu 10.12.2020. https://www.redblobgames.com/maps/terrain-from-noise/

Gustavson S. 2005. Simplex noise demystified. http://staffwww.itn.liu.se/~stegu/simplexnoise/simplexnoise.pdf