KARELIA-AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutusohjelma

Samu Nykänen

VR-SOVELLUKSEN OPTIMOINTI JA METSÄN VISUALISOINTI UNREAL ENGINE 4 -PELIMOOTTORILLA

Opinnäytetyö
Marraskuu 2018

Karelia AMMATTIKORKEAKOULU	**OPINNÄYTETYÖ** **Marraskuu 2018** **Tietojenkäsittelyn koulutus** Tikkarinne 9 80200 JOENSUU +358 13 260 600 (vaihde)

Tekijä(t)
Samu Nykänen

Nimeke
VR-sovelluksen optimointi ja metsän visualisointi Unreal Engine 4 -pelimoottorilla

Toimeksiantaja
Joensuu Games OSK

Tiivistelmä

Opinnäytetyö sisältää kaksi työprojektia, joissa teemana oli metsän visualisointi. Ensimmäisessä projektissa tavoitteena oli optimoida metsäalue virtuaalitodellisuutta varten Unreal Engine 4 -pelimoottorilla. Projektin tavoitteena oli päästä suorituskyvyssä 90 kuvaan sekunnissa. Ensimmäisen projektin raportissa avataan optimoinnissa esille tulevia termejä sekä kuvataan Unreal Engine 4 -pelimoottorin tarjoamia työkaluja suorituskyvyn mittaukseen. Lisäksi kuvataan projektin aikana tehtyjä optimointeja ja niiden vaikutusta.

Toisen projektin tavoitteena oli tehdä neliökilometrin laajuisen japanilaisen metsäalueen visualisointi asiakkaan antamien tietojen perusteella. Raportissa kerrotaan, mitä aineistoa asiakas toimitti meille ja kuinka sitä hyödynnettiin Unreal Engine 4 -pelimoottorissa. Raportissa kuvataan myös, mitä ongelmia Unreal Engine 4 -pelimoottorin omien puunistutustyökalujen käyttö aiheutti. Lisäksi kerrotaan, kuinka toteutin oman työkalun puun istuttamista varten, mikä hyödynsi asiakkaan antamia puutietoja.

Kieli	Sivuja 46
suomi	

Asiasanat

virtuaalitodellisuus, optimointi, metsän visualisointi

Karelia
UNIVERSITY OF APPLIED SCIENCES

THESIS
November 2018
Degree Programme in business IT

Tikkarinne 9
80200 JOENSUU
FINLAND
+ 358 13 260 600 (switchboard)

Author (s)
Samu Nykänen

Title
Optimization for virtual reality and forest visualization in Unreal Engine 4

Commissioned by
Joensuu Games COOP

Abstract

The thesis includes two work projects with common theme being forest visualization. Goal of the first project was to optimize forest area for virtual reality with the Unreal Engine 4. Performance goal of the project was 90 frames per second. Report of the first project explains frequently used terms on game optimization and describes the tools provided by the Unreal Engine 4 for performance monitoring. It also describes the optimizations made during the project and their impact on performance.

The goal of the second project was to visualize 1 * 1 Km area of japanese forest based on the information provided by the customer. Report explains what data customer provided to us and how they were used in the Unreal Engine 4. Report also describes the problems caused by tree planting tools included in Unreal Engine 4. In addition report explains how I implemented my own tool for tree planting, which makes use of the tree data provided by the customer.

Language

Finnish

Pages 46

Keywords

virtual reality, optimization, forest visualization

Sisältö

1 Johdanto

Opinnäytetyö sisältää kaksi työprojektia, joissa olin mukana työskennellessäni Joensuu Games OSK:ssa. Molemmissa projekteissa yhteinen teema on metsän visualisointi. Ensimmäinen projekti oli metsäalueen optimointi virtuaalitodellisuutta varten. Tässä osassa vastataan seuraaviin kysymyksiin: Mitkä asiat vaikuttavat suorituskykyyn optimoitavassa alueessa? Mitä erityistä pitää ottaa huomioon virtuaalitodellisuussovelluksessa suorituskyvyn suhteen? Miten prosessorin, näytönohjaimen ja muistin suorituskykyä mitataan Unreal Engine 4 -pelimoottorilla? Unreal Engine 4 -pelimoottori oli tullut tutuksi ennen tätä projektia, mutta optimoinnista ja suorituskyvyn mittaamisesta ei ollut aikaisempaa osaamista millään pelimoottorilla. Tämä oli myös ensimmäinen virtuaalitodellisuussovellus, jossa olen ollut mukana

Toinen projekti oli japanilaisen metsäalueen visualisointi. Tässä osassa kuvataan, kuinka asiakkaalta saatujen tietojen perusteella toteutettiin visualisoitava metsäalue. Tämä osa vastaa seuraaviin kysymyksiin: Kuinka hyödynnetään oikean maailman korkeusmalleja Unreal Engine 4 -pelimoottorissa? Kuinka oikean maailman puusto tietoja hyödynnetään puiden istuttamissa Unreal Engine 4 -pelimoottorissa? Metsän visualisoinnista minulla oli aikaisempaa kokemusta muista projekteista, mutta tämän oli ensimmäinen, jossa oli tarkat puustotiedot toimitettuna. Tiedoista näki esimerkiksi yksittäisen puun X- ja Y-koordinaatin. Tarkoitus oli löytää keino, jolla pystyisimme hyödyntämään koordinaatteja puiden istutuksessa Unreal Engine 4 -pelimoottorilla.

2 Case virtuaalitodellisuussovelluksen optimointi

Optimoitava sovellus on virtuaalitodellisuussovellus, jossa käyttäjää liikutetaan ennalta määrätyn reitin mukaisesti pienellä metsäalueella. Tässä tapauksessa tarkoitus ei ollut tehdä visualisaatiota oikeasta metsäalueesta,

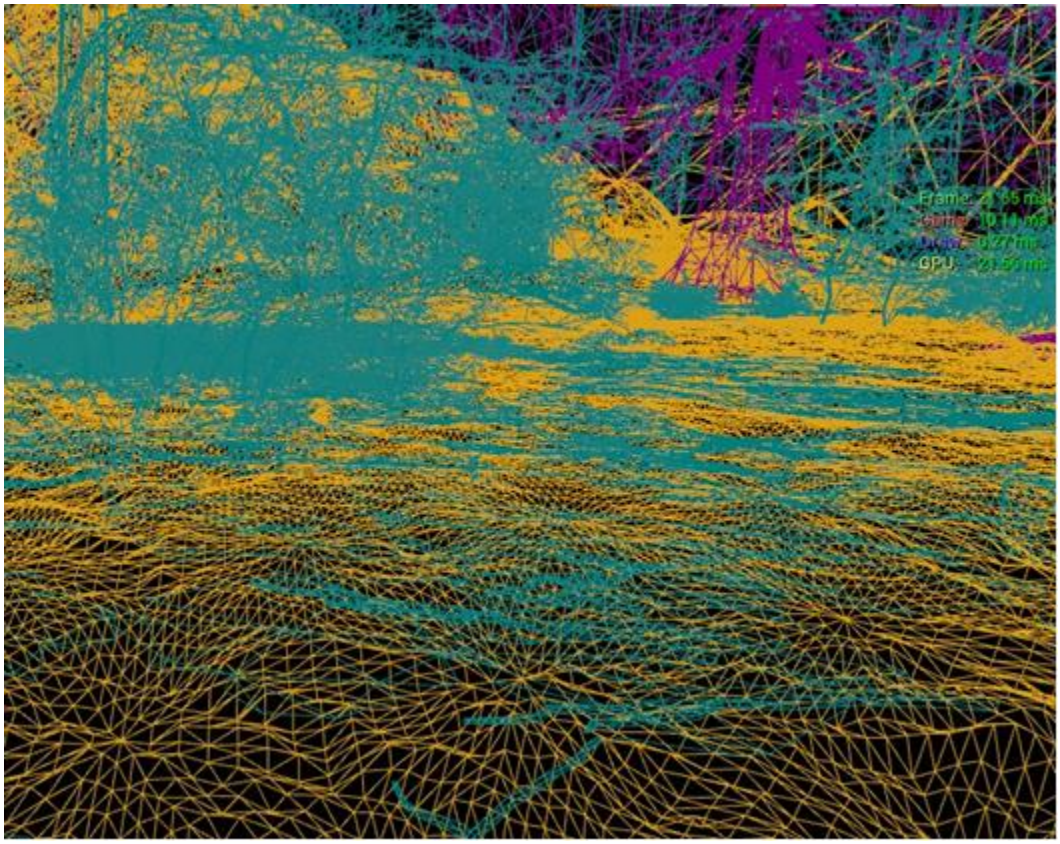
vaan kyseessä oli kuvitteellinen metsä. Sovellusta on tarkoitus esitellä erilaisissa tapahtumissa kuten tekniikkamessuilla ja toimia siten markkinointityökaluna toimeksiantajalle. Toimeksiantajana on Joensuu Games OSK, joka tekee mm. visualisointeja metsistä ja kaupungeista. Sovellus tehtiin Oculus Rift -virtuaalilaseille. Tavoitteena oli sovellus parhaalla mahdollisella kuvanlaadulla, jonka suorituskyky olisi 90 kuvaa sekunnissa.

2.1 Termit

Selkeyden vuoksi tässä luvussa kuvataan lyhyesti opinnäytetyössä esille tulevia termejä.

2.1.1 Kolmio

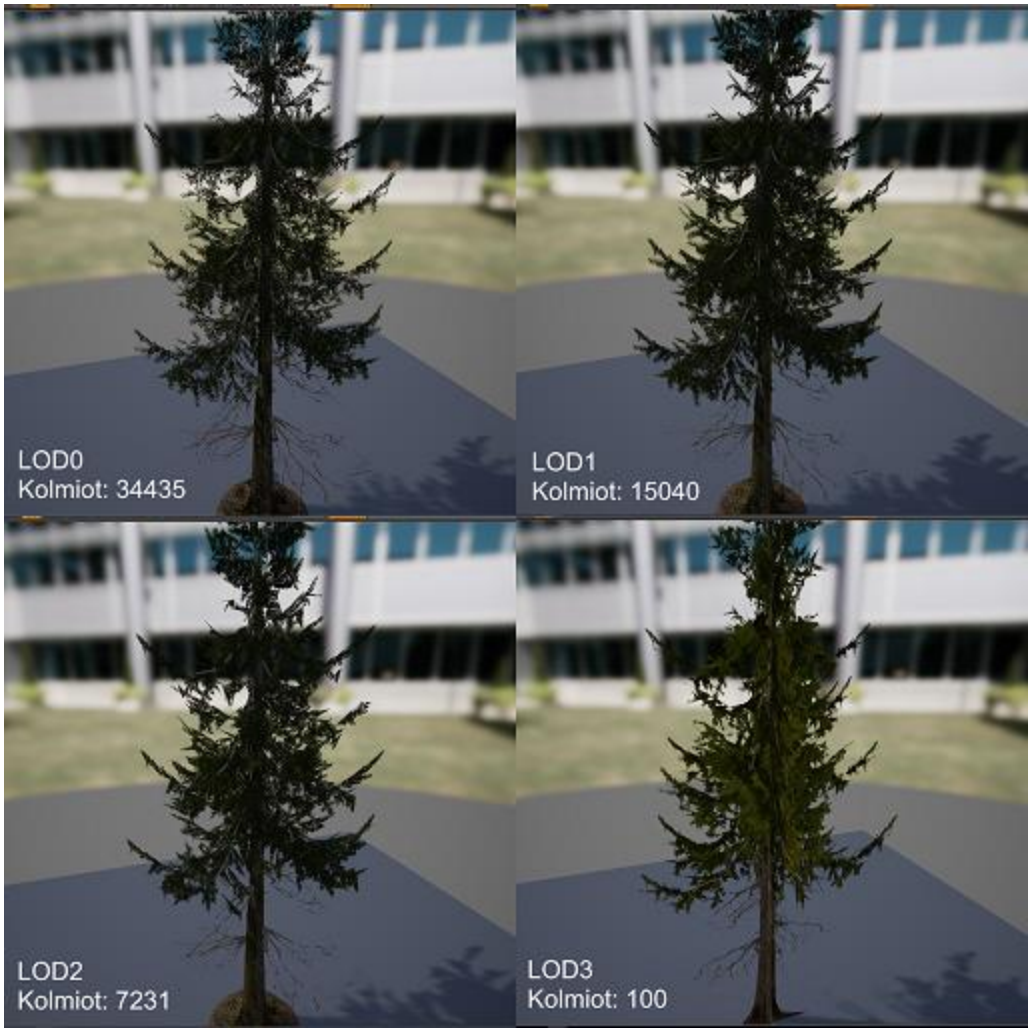
3D-mallit renderöidään Unreal Engine 4 -pelimoottorissa kolmioiden avulla (Epic Games 2018a). Mitä enemmän ja mitä pienempiä kolmiot ovat, niin sitä tarkempia 3D-malleja voidaan piirtää. Suuret kolmiomäärät kuitenkin heikentävät suorituskykyä, sillä jokainen kolmio joudutaan käsittelemään kuvan piirtämisen yhteydessä. Kuvassa 1 kolmiot näkyvät selkeästi.



Kuva 1. Unreal Engine 4 -pelimoottorin rautalankanäkymä.

2.1.2 Tarkkuustaso

Tarkkuustaso (level of detail) on optimointitapa, jossa peli vaihtaa peliobjektin 3D-mallia yksinkertaisemmaksi riippuen objektin etäisyydestä pelaajaan tai kameraan. Tarkkuustaso on erittäin hyödyllinen, koska sen avulla objektit voi pitää yksityiskohtaisina läheltä katsottuna, mutta kauemmaksi siirryttäessä objektin malli voidaan vaihtaa, joko yksinkertaisemmaksi ja kolmiomäärältään pienempään tai monimutkaisemmaksi ja kolmiomäärältään suurempaan. (Epic Games 2017a.) Tämä vähentää piirrettävien kolmioiden määrää ja siten parantaa suorituskykyä. Kuvassa 2 verrataan sovelluksessa käytetyn puumallin tarkkuustasoja. Kuvassa 2 LOD0 on monimutkaisin puun 3D-malli, jossa on 34 435 kolmioita. LOD3 on taas saman puun yksinkertaisin 3D-malli.



Kuva 2. Esimerkki yhden sovelluksessa käytettävän puun tarkkuustasoista.

2.1.3 Reunanpehmennys

Koska kuva piirretään pikseleinä tietokoneen ruudulle, niin objektien reunat voivat näyttää ns. sahalaitaisina. Pikselit ovat neliön muotoisia, joten sahalaitaisuus esiintyy reunoilla, jotka menevät vinottain. Pystysuoraan tai vaakasuoraan menevillä reunoilla sahalaitaisuutta ei esiinny. Reunanpehmennys säätää reunojen lähellä olevien pikseleiden väriä, jolloin saadaan reuna näyttämään suoralta. Haittavaikutuksena saattaa olla hieman sumeampi kuva. (McKee 2017.)

Unreal Engine 4 -pelimoottori tarjoaa kolme reunanpehmennystekniikkaa: Multisample anti-aliasing eli MSAA, Fast approximate anti-aliasing eli FXAA ja Temporal anti-aliasing.

MSAA ottaa näytteitä pikseleistä ja niiden perusteella päättää, mitkä pikselit kuuluvat piirrettävän kolmion alueelle. Kolmion reunoilla olevien pikseleiden väriä säädetään sen mukaan, montako näytettä pikselistä oli kolmion alueella. Mikäli näytteitä otetaan esimerkiksi kaksi ja toinen näyte on kolmion sisällä ja toinen ei, niin pikselin väri on keskiarvo kolmion ja sen viereisen pikselin välillä. Oletuksena Unreal Engine 4 -pelimoottorissa MSAA ottaa 4 näytettä. MSAA ei pehmennä tekstuureja, koska se tehdään ennen pikselin varjostusta. (Pettineo 2016.)

FXAA käy läpi kaikki pikselit piirrettävästä kuvasta ja tunnistaa reunat käyttämällä esimerkiksi pikselin kirkkautta. Samaan reunaan kuuluvilla pikseleillä on suurin piirtein sama kirkkaus. Kun reunat ovat tunnistettu, niin FXAA sumentaa ne säätämällä reunoilla olevien pikseleiden väriä. Tämä tekniikka on huomattavasti nopeampi kuin MSAA. Sovelluksessa FXAA hidasti suorituskykyä noin yhden millisekunnin. MSAA hidasti suorituskykyä 8 millisekuntia. FXAA:n haittavaikutuksena on sumeampi kuva varsinkin pienemmillä resoluutioilla. (Atwood 2011.)

TemporalAA pehmentää reunoja yhdistämällä MSAA-tyylistä näytteiden ottoa pikseleistä sekä kuvan FXAA-tyylistä reunojen sumentamista. Tämän lisäksi temporalAA pehmentää liikettä. (Nvidia 2017.) Kahteen edelliseen tekniikkaan verrattuna temporalAA sumentaa kuvaa virtuaalitodellisuudessa eniten. Taulukossa 1 verrataan eri reunanpehmennystekniikoiden vaikutusta suorituskykyyn.

Taulukko 1. Reunanpehmennystekniikoiden vaikutus suorituskykyyn.

AA tekniikka	**Frametime**
Ei mitään	17.76 ms
MSAA	25.89 ms

FXAA	18.89 ms
TemporalAA	21.99 ms

Pelkästään suorituskyvyn kannalta järkevin vaihtoehto olisi FXAA, mutta MSAA tekee kuvasta huomattavasti terävämmän virtuaalitodellisuudessa, joten alustavasti valitsin sen reunanpehmennystekniikaksi.

2.1.4 Piirtokutsu

Ennen kuin mitään piirretään ruudulle prosessori tarkastaa ensiksi jokaisen objektin osalta, tarvitseeko sitä edes piirtää. Se voi jäädä jonkin toisen objektin taakse ja siinä tapauksessa sitä ei piirretä. Tätä tekniikka kutsutaan nimellä occlusion culling. Tämän lisäksi prosessori kerää tiedon, miten objekti piirretään eli minkälainen materiaali objektilla on ja miten valaistus vaikuttaa siihen. Kun prosessori on käsitellyt kaiken tarvittavan, se lähettää piirtokutsut näytönohjaimelle, joka lopulta piirtää kuvan annettujen ohjeiden mukaan ruudulle. Koska prosessori käsittelee jokaisen objektin, niin jokainen erillinen objekti on aina oma piirtokutsunsa. Mikäli objektissa on useampi materiaali, niin piirtokutsujen määrä nousee. Otetaan esimerkkinä puu, jossa on omat materiaalinsa rungolle, oksille ja lehdille. Tällöin prosessori joutuu kertomaan näytönohjaimelle erikseen, mille puun osalle tulee mikäkin materiaali. (Schreibt 2015.)

Optimoinnin kannalta piirtokutsujen määrä kannattaa pyrkiä pitämään mahdollisimman matalana. Usein näytönohjain ennättää toteuttaa oman osansa kuvan piirtämisestä nopeammin kuin prosessori. Tällöin näytönohjain joutuu odottelemaan seuraavan kuvan piirtokutsuja prosessorilta. Piirtokutsujen vähentämisellä voidaan ehkäistä tätä tilannetta. (Jukić 2015.)

2.1.5 Varjostin

Varjostin on pieni ohjelma, jolla 3D-mallia voidaan muokata renderöinnin yhteydessä. Tyypillisimmät varjostimet ovat verteksivarjostin, pikselivarjostin ja geometriavarjostin. Verteksivarjostimella voidaan muokata mm. 3D-mallin kolmioiden kärkipisteiden, eli verteksien, sijaintia. Pikselivarjostimella käsitellään mallin yksittäisen pikselin ominaisuuksia, kuten väriä. Geometriavarjostimella voidaan lisätä kolmioita ja verteksejä 3D-mallille. (Wikipedia 2017a)

2.1.6 Renderöijä

Renderöijä muuttaa 3D-mallit kuviksi ja piirtää ne ruudulle (Wikipedia 2017b). Unreal Engine 4 -pelimoottorissa on kaksi eri renderöijävaihtoehtoa: deferred renderer ja forward renderer. Ensimmäinen odottaa, kunnes kaikki 3D-mallin geometria on mennyt varjostimien läpi ja lisää sitten valaistuksen malliin. Jälkimmäisessä erillistä valaistusvaihetta ei ole.

Deferred renderöijässä pikselin tiedot, kuten esimerkiksi väri, kiiltävyys, varastoidaan GBuffereihin. Kun jokaisen pikselin tiedot ovat varastoitu, tehdään erillinen valaistusvaihe. (Owens 2013.)

Forward rendering voi olla nopeampi kuin deferred renderer ja se mahdollistaa myös MSAA:n käytön. (Epic Games 2017b.)

Taulukko 2. Renderöijän vaikutus suorituskykyyn.

Renderöijä	**Frametime**
Deferred	21.68 ms
Forward	17.76 ms

Taulukosta 2 nähdään, että optimoinnin kannalta selkeästi parempi valinta tähän sovellukseen on forward renderöijä. Testi on tehty stat unitgraph -komennolla sovelluksen ollessa käynnissä. Molemmat tulokset ovat otettu

samasta kohtaa, joka on varmistettu asettamalla pelaajan aloituspiste pelimaailmaan. Molemmissa tapauksissa reunanpehmennys ei ollut käytössä.

2.2 Optimointi

Tässä luvussa kuvataan, mitä ovat optimointi ja profilointi. Lisäksi käydään läpi Unreal Engine 4 -pelimoottorin tarjoamia profilointityökaluja ja niiden tulkintaa.

2.2.1 Yleiskuvaus

Yleisesti optimointi tarkoittaa pelkistetysti täysin saman asian tekemistä vähemmällä vaivalla. Tämä määrittely ei kuitenkaan täysin sovellu pelialalle. Parempi tapa määritellä pelioptimointi on, että sama peli toimii yhtä tehokkaasti usealla eri alustalla (Thoman 2016).

Optimointia tehdään yleensä kolmessa kategoriassa, prosessori, näytönohjain ja muisti. Prosessorin suorituskykyyn peleissä vaikuttaa pelilogiikka eli koodi. Näytönohjain tekee itse kuvien piirtämisen ruudulle ja tämän takia sen suorituskykyyn vaikuttaa moni asia.

Muistin kulutus voidaan jakaa kolmeen osaan: RAM eli keskusmuisti, VRAM eli videomuisti ja itse sovelluksen viemä tila kovalevyltä. Näistä sovelluksen suorituskyvyn kannalta tärkein on videomuisti. Sen kokoon vaikuttavat mm. käytössä olevien tekstuurien resoluutio, sovelluksen resoluutio ja piirrettävien objektien määrä ruudulla. Sovelluksen koko kovalevyllä ei vaikuta varsinaisesti suorituskykyyn.

2.2.2 Profilointi

Jotta voidaan lähteä optimoimaan, niin ensiksi pitää saada selville, mitkä asiat sovelluksessa vaativat optimointia. Tämä tapahtuu mittaamalla sovelluksen

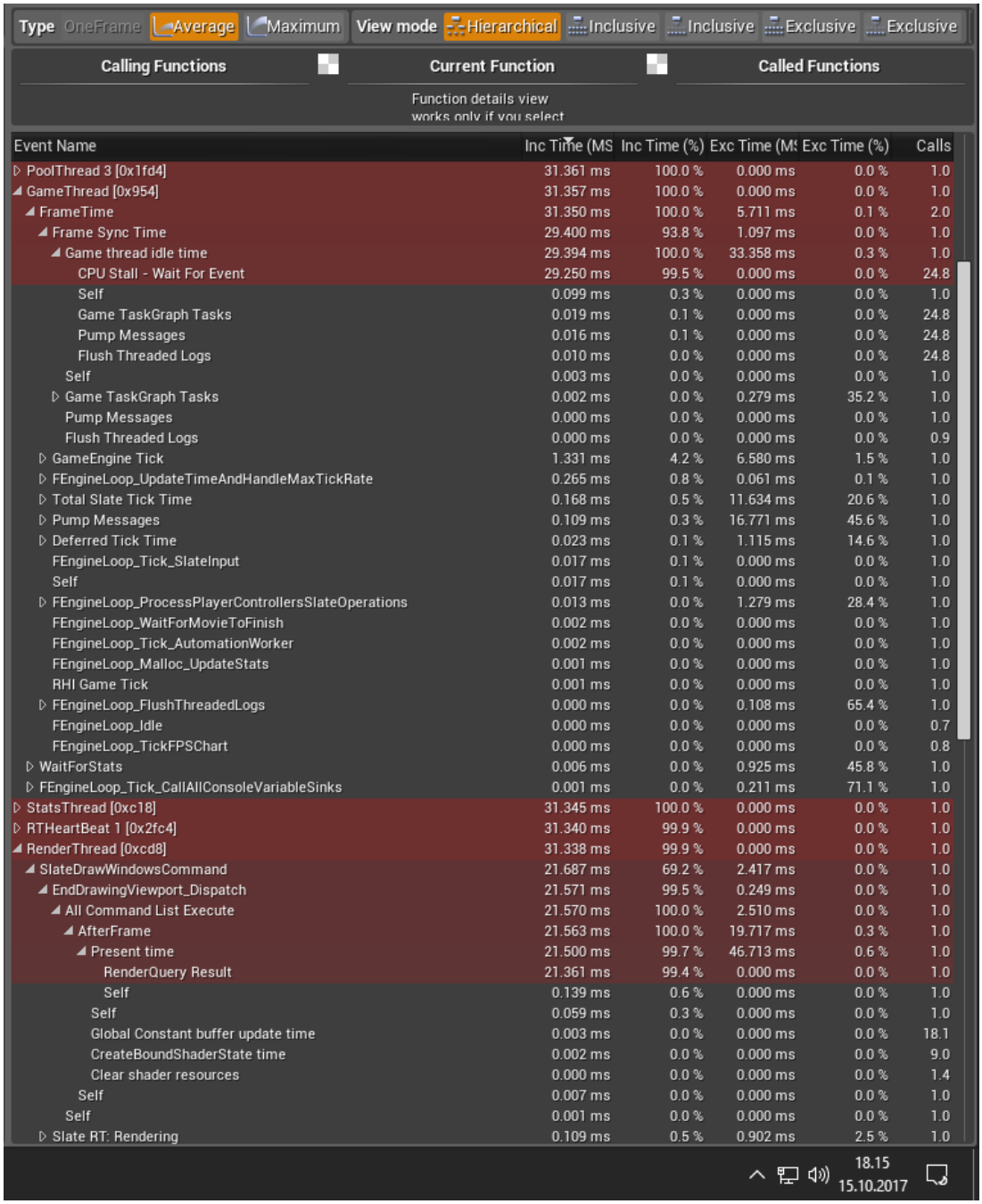
suorituskykyä prosessorin, näytönohjaimen ja muistin osalta (Epic Games 2018b). Tätä vaihetta kutsutaan profiloinniksi.

Pelien suorituskyky ilmoitetaan yleensä kuvina sekunnissa eli FPS (Klappenbach 2018). Tarkempi tapa mitata on kuitenkin yhden kuvan piirtämiseen kulunut aika millisekunteina eli frametime. Unreal Engine 4 -pelimoottorin profilointityökalut käyttävät tätä mittaria, kuten esimerkiksi stat unit-komento, joka erittelee prosessorin ja näytönohjaimen käyttämän ajan kuvan piirtämiseen. Tämä on hyödyllinen antamaan suuntaa, että kumpaa pitää lähteä tarkemmin profiloimaan.

Oculus Riftin kanssa pitää muistaa, että mikäli FPS ei pääse 90:een kuvaan sekunnissa, niin se putoaa 45:een kuvaan sekunnissa. Tämä tekee profiloinnista melkein mahdotonta, koska sovelluksen todellista suorituskykyä ei voi tietää. Se voi olla esimerkiksi 60 FPS tai 70 FPS, mutta Unreal Engine 4 -pelimoottorin profilointityökalut ilmoittavat tässä tapauksessa 45 FPS. Ratkaisu tähän on emuloida virtuaalitodellisuutta, jolloin nähdään sovelluksen todellinen suorituskyky. (Epic Games 2017c.)

2.2.3 Prosessorin profilointi

Tarkan profiloinnin voi tehdä stat startfile ja stat stopfile-konsolikomennoilla. Startfile-komento aloittaa tiedon keräämisen prosessorin suorituskyvystä erilliseen lokitiedostoon, jonka voi avata Unreal Engine 4 -pelimoottorin session frontend-työkalulla. Stopfile taas pysäyttää tiedon tallentamisen. (Tellez 2014.)



Kuva 3. Unreal Engine 4 -pelimoottorin session frontend-näkymä.

Kuvassa 3 nähdään tarkka prosessorin profilointinäkymä. Asiat, joihin kannattaa kiinnittää huomiota prosessorin profiloinnin osalta on Game Thread ja Render Thread. Game Threadin kohdalla oleva arvo tarkoittaa sovelluksen pelilogiikan viemää aikaa millisekunteina. Tämä sisältää mm. sovelluksessa olevan koodin. Render Tread valmistelee objektien piirtämistä näytönohjainta varten, kuten esimerkiksi valaistuslaskennat. (Epic Games 2017d.) Kuvassa 3 nähdään, että suurin osa Game Threadin ajasta menee odotteluun. Render

Threadin osalta tapa millä profiloinnin tein, eli emuloimalla virtuaalitodellisuutta, antoi väärän kuvan todellisesta tilanteesta. Kuvassa 3 nähdään, että suurin osa Render Threadin ajasta kuluu SlateDrawWindowsCommand-suoritukseen. Slate on Unreal Engine 4 -pelimoottorin käyttöliittymä framework, jolla itse editorin käyttöliittymä on tehty (Epic Games 2018c). Optimoitavassa sovelluksessa ei kuitenkaan ole käyttöliittymää, joten sen valmisteluun ei pitäisi kulua aikaa. Itse Oculus Riftillä mitatessa Render Threadin viemä aika oli huomattavasti pienempi. Tästä voidaan päätellä, että tarvetta prosessori optimoinnille sovelluksessa ei ole.

2.2.4 Näytönohjaimen profilointi

Sovelluksen tarkempi näytönohjaimen profilointi on tehty profileGPU-komennolla, joka kerää yhden kuvan piirtämiseen ruudulle kuluneen ajan ja tallentaa sen lokitiedostoon. Komento on ajettu sovelluksen sisällä samasta paikasta ja katsomalla aina samaan suuntaa, jotta tuloksia voidaan verrata helposti keskenään. Tämä on varmistettu määrittämällä paikka, jonne pelaaja luodaan. Tällöin pelaajan sijainti ja suunta on aina sama, kun sovellus käynnistetään. (Epic Games 2017e.)

Taulukko 3. profileGPU-komennon tuottama lokitiedosto sovelluksen alkutilanteesta.

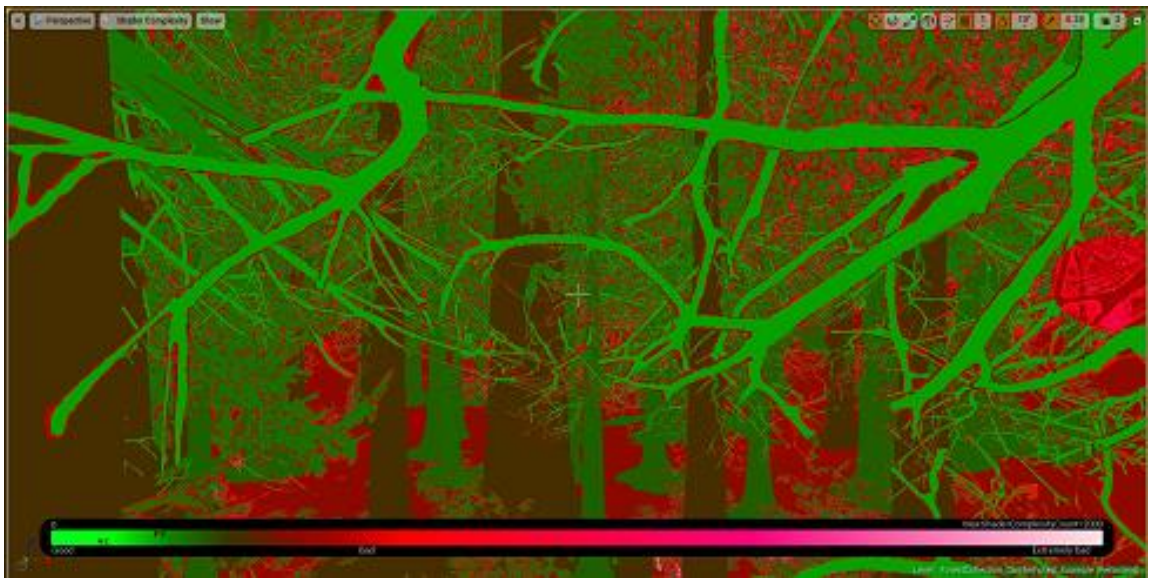
```
100.0% 35.91ms   FRAME 3302 draws 12608295 prims 13351365 verts
99.4% 35.71ms   Scene 3302 draws 12608295 prims 13351365 verts
LogRHI: 8.2% 2.94ms   PrePass DDM_AllOpaque (Forced by DBuffer)
LogRHI: 0.3% 0.11ms   BeginOcclusionTests
LogRHI: 0.1% 0.04ms   HZB SetupMip 0 1024x1024
LogRHI: 0.2% 0.09ms   HZB SetupMips Mips:1..9 512x512
LogRHI: 0.1% 0.04ms   HZB SetupMip 0 1024x1024
LogRHI: 0.1% 0.04ms   HZB SetupMips Mips:1..9 512x512
LogRHI: 7.5% 2.68ms   ShadowDepths
LogRHI: 0.3% 0.11ms   ComputeLightGrid
```

```
LogRHI: 7.4% 2.67ms   VolumetricFog
LogRHI: 7.1% 2.54ms   VolumetricFog
LogRHI: 0.1% 0.05ms   BeginRenderingSceneColor
LogRHI: 21.8% 7.81ms   BasePass
LogRHI: 0.0% 0.01ms   ClearStencilFromBasePass
LogRHI: 0.2% 0.05ms   ClearTranslucentVolumeLighting
LogRHI: 5.6% 2.01ms   Lights
LogRHI: 0.1% 0.03ms   InjectAmbientCubemapTranslucentVolumeLighting
LogRHI: 0.5% 0.17ms   FilterTranslucentVolume 64x64x64 Cascades
LogRHI: 2.6% 0.92ms   SkyLightDiffuse
LogRHI: 1.1% 0.39ms   ScreenSpaceReflections 1756x1976
LogRHI: 2.5% 0.90ms   ReflectionEnvironment ComputeShader 1756x1976 Tile:8x8
LogRHI: 1.1% 0.40ms   ScreenSpaceReflections 1755x1976
LogRHI: 4.8% 1.71ms   ReflectionEnvironment ComputeShader 1755x1976 Tile:8x8
LogRHI: 1.0% 0.37ms   Atmosphere 1756x1976
LogRHI: 0.9% 0.33ms   Atmosphere 1755x1976
LogRHI: 0.8% 0.29ms   ExponentialHeightFog 1756x1976
LogRHI: 0.9% 0.31ms   ExponentialHeightFog 1755x1976
LogRHI: 0.0% 0.01ms   Translucency
LogRHI: 8.9% 3.19ms   PostProcessing
```

Taulukosta 3 nähdään, että eniten aikaa vievät vaiheet ovat BasePass, VolumetricFog, PostProcessing ja PrePass. BasePass valmistelee tietoja muita vaiheita varten. Esimerkiksi käy läpi objektin materiaalin värin, kiiltävyyden jne. ja varastoi ne GBuffereihin. Forward renderöijässä kaikki valaistukset ja varjostukset pikselille tehdään myös tässä. Tämä selittää, miksi myöhemmässä vaiheessa profilointia tämän vaiheen viemä aika nousee merkittävästi, koska forward rendering on otettu käyttöön. Volumetric fog on visuaalinen efekti, jolla saadaan auringon valoon näkyvät valokeilat. PostProcess sisältää muita visuaalisia efektejä kuten bloom, motion blur ja tässä tapauksessa myös reunanpehmennyksen. PrePass vaiheessa kerätään mm. objektien syvyystiedot eli tarkistetaan jääkö jokin objekti toisen objektin taakse, jolloin takana olevaa objektia ei tarvitse piirtää. (Tech Art Aid 2017.)

BasePass-aikaa voi parhaiten parantaa yksinkertaistamalla objektien materiaaleja, pienentämällä sovelluksen resoluutiota tai vähentämällä piirtokutsuja ja kolmiota. PrePass-ajan parantamiseen toimii myös piirtokutsujen ja kolmioiden vähentäminen. (Tech Art Aid 2017.) Iso parannus tulee ottamalla volumetric fog kokonaan pois, mutta se parantaa sovelluksen visuaalista ilmettä paljon, joten pyrin pitämään sen päällä, jos vain mahdollista. Muita pieniä parannuksia saadaan ottamalla pois joitakin post process-efektejä, kuten bloom ja motion blur. Nämä eivät muutenkaan näytä kovin hyvältä virtuaalitodellisuudessa. Lisäksi dynaamisten varjojen piirtoetäisyyttä rajoittamalla saadaan suorituskykyä paremmaksi.

Objektien materiaalien yksinkertaistamista helpottaa Unreal Engine 4 -pelimoottorin varjostimien monimutkaisuusnäkymä. Tämä näkymä näyttää, kuinka monta varjostinkäskyä yksi pikseli tarvitsee, jotta se voidaan piirtää ruudulle. Käskyjen määrää riippuu, miten objektin materiaali on Unreal Engine 4 -pelimoottorin materiaali editorilla tehty. (Epic Games 2018d.)



Kuva 4. Varjostimien monimutkaisuusnäkymä.

Kuvassa 4 vihreä väri tarkoittaa vähäistä käskyjen määrää, kun taas valkoinen tarkoittaa suurta käskyjen määrää. Tässä näkymässä pitää ottaa kuitenkin huomioon, että jotkin käskyt ovat hitaampia toteuttaa kuin toiset ja varjostimien monimutkaisuusnäkymä ei huomioi sitä (Epic Games 2017f).

2.2.5 Muistin profilointi

Kattavan raportin video- ja keskusmuistin kulutuksesta saa Unreal Engine 4 -pelimoottorilla memreport-konsolikomennolla. Muistiraportti listaa kaikki muistia vievät asiat, kuten objektit pelimaailmassa. Lisäksi raportista näkee esimerkiksi koodissa olevien muuttujien viemä muisti. (Zeigler 2014.) Kuvissa 5 ja 6 näkyvät oleellisimmat tiedot muistiraportin sisällöstä. Kuvassa 5 sovelluksen käyttämä muisti näkyy kohdissa process physical memory ja process virtual memory. Kaksi muuta kohtaa näyttävät kaikkien käynnissä olevien sovellusten käyttämän muistin.

```
Platform Memory Stats for WindowsNoEditor
Process Physical Memory: 659.82 MB used, 903.58 MB peak
Process Virtual Memory: 2727.48 MB used, 2773.36 MB peak
Physical Memory: 12319.16 MB used,  28509.32 MB free, 40828.48 MB total
Virtual Memory: 3190.75 MB used,  28509.32 MB free, 134217728.00 MB total
```

Kuva 5. Alkutilanne sovelluksen viemästä keskusmuistista.

```
RHI resource memory (not tracked by our allocator)
        2228160  -  Render target memory Cube - STAT_RenderTargetMemoryCube - STATGROUP_RHI - STATCAT_Advanced
      346578944  -  Render target memory 3D - STAT_RenderTargetMemory3D - STATGROUP_RHI - STATCAT_Advanced
        1353376  -  Uniform buffer memory - STAT_UniformBufferMemory - STATGROUP_RHI - STATCAT_Advanced
      870651756  -  Render target memory 2D - STAT_RenderTargetMemory2D - STATGROUP_RHI - STATCAT_Advanced
         540676  -  Texture memory 3D - STAT_TextureMemory3D - STATGROUP_RHI - STATCAT_Advanced
      877120688  -  Texture memory 2D - STAT_TextureMemory2D - STATGROUP_RHI - STATCAT_Advanced
       61996644  -  Texture memory Cube - STAT_TextureMemoryCube - STATGROUP_RHI - STATCAT_Advanced
       98334980  -  Vertex buffer memory - STAT_VertexBufferMemory - STATGROUP_RHI - STATCAT_Advanced
       51577462  -  Index buffer memory - STAT_IndexBufferMemory - STATGROUP_RHI - STATCAT_Advanced
2203.353MB total
```

Kuva 6. Alkutilanne videomuistin kulutuksesta.

Raportin pohjalta voidaan todeta, että video- tai keskusmuistin kulutus on vähäinen, joten muistin kulutusta ei ole tarvetta optimoida, sillä kohdetietokoneessa on keskusmuistia 40 gigatavua ja videomuistia 8 gigatavua. Sovellus tarvitsee n. 658 megatavua keskusmuistia ja n. 2 gigatavua videomuistia. Muistin kulutusta kuitenkin seurataan optimoinnin edetessä.

2.3 Toteutus

Tämä projekti ei mennyt asiakkaalle, vaan kyseessä oli markkinointia varten tehty sovellus. Tästä syystä projektissa ei ollut aluksi muita tekijöitä kuin minä. Graafikon puuttuminen tarkoitti sitä, että kaikki 3D-mallit, tekstuurit materiaalit jne. piti olla valmiita. Päädyimme käyttämään projektin pohjana Unreal kauppapaikasta aiemmin ostetun puupaketin esimerkkikenttää. puupaketti oli ostettu aikaisempaa projektia varten, jossa sitä ei kuitenkaan hyödynnetty. Puupaketti piti sisällään 16 erilaista puu 3D-mallia. Lisäksi puiden materiaalit ja tekstuurit tulivat mukana. Optimointi haaste oli suuri, sillä kyseisessä kentässä oli paljon suorituskyvyllisesti raskaita visuaalisia efektejä käytössä. Esimerkkejä tästä mm. ovat dynaamiset varjot, volumetrinen sumu, TemporalAA ja tiheä aluskasvillisuus. Dynaamiset varjot piirretään joka kuvalla uudestaan. Lisäksi objektit, jotka tekevät dynaamisia varjoja joudutaan piirtämään kahdesta suunnasta, pelaajan kamerasta sekä kentässä olevasta valonlähteestä. Tämä lisää kolmiomäärää. Vaihtoehto tähän olisi ollut staattiset varjot, joita ei tarvitse piirtää uudestaan jokaisella kuvalla. (Epic Games, 2017g.) Tässä tulee vastaan yksi ongelma, jonka takia en päätynyt käyttämään staattisia varjoja. Liikkuvien objektien, kuten esimerkiksi puiden oksien, varjot pysyisivät staattisilla varjoilla paikoillaan.

Taulukossa 4 on merkattu suorituskyky, piirtokutsut sekä kolmimäärät ennen optimointia. Kuvassa 7 vuorostaan näkyy miltä puu paketin esimerkki kenttä näytti ennen optimointia.

Taulukko 4. Suorituskyky ennen optimointia.

Suorituskyky	35,91 ms
Piirtokutsuja	3302 kpl
Kolmioita	12 608 295 kpl
Tavoite suorituskyky	11,1 ms

Kuva 7. Alkutilanne.

Ensimmäinen askel oli muuttaa projektin asetuksia sopimaan paremmin virtuaalitodellisuuteen. Oletuksena Unreal Engine 4 -pelimoottorissa on käytössä FPS katto, jonka maksimiarvo on 62 FPS. Tämä ei tietenkään riitä 90 FPS:n saavuttamiseksi, joten otin FPS katon kokonaan pois. Lisäksi Epic Games suosittelee virtuaalitodellisuudessa ottamaan Depth of fieldin ja motion blurin pois käytöstä. Molemmat voivat aiheuttaa pahoinvointia. (Epic Games, 2017h.)

Tämän jälkeen selvitin, että onko suorituskyvyn pullonkaulana prosessori vaiko näytönohjain. Tämä tapahtui nopeasti unit graph-komennolla, joka näyttää eroteltuna yhden kuvan piirtämiseen kuluneen ajan. Tämä komento näytti, että prosessorin viemä aika oli n. 2 millisekuntia, joten selkeästi pullonkaulana tässä projektissa on näytönohjain. Kaikki optimoinnit ovat tästä syystä grafiikkaan liittyviä, eikä koodiin.

Jokainen puu oli oma 3D-mallinsa, joten jokaisesta puusta tuli vähintään yksi piirtokutsu. Tästä johtuen ensimmäinen askel oli poistaa kaikki puut ja korvata ne Unreal Engine 4 -pelimoottorin kasvityökalulla istutetuilla puilla. Näin Unreal Engine 4 -pelimoottori osaa niputtaa usean puun yhteen piirtokutsuun. Jotta

suorituskyky tuloksia voisi verrata keskenään, uudet puut piti istuttaa samoille paikoille kuin jo olemassa olevat. Tämä oli työläin vaihe koko projektissa, sillä tein kaiken käsin. Istutin yhden puun ja siirsin sen manuaalisesti samaan kohtaan kuin olemassa oleva, jonka jälkeen poistin alkuperäisen puun kentästä. Puita kentässä oli 687 kappaletta.

Taulukko 5. Suorituskyky puiden uudelleen istuttamisen jälkeen.

Suorituskyky	38,51 ms
Piirtokutsuja	2409 kpl
Kolmioita	15 644 292 kpl
Tavoite suorituskyky	11,1 ms

Oletin, että saisin tällä parannettua suorituskykyä merkittävästi, mutta kuten taulukosta 5 nähdään, onnistuin vain huonontamaan sitä 2,6 millisekunnilla. Syy tähän oli se, että päädyin pienentämään uudelleen istutettavien puiden kokoa. Aikaisemmin suuremmat puut peittivät enemmän niiden takana olevia objekteja, joten kolmio määrä oli myös pienempi. Toisin sanoen vähensin piirtokutsuja, mutta lisäsin piirrettävien kolmioiden määrää n. 12 miljoonasta 15 miljoonaan. Päädyin pienentämään puut, koska ne näyttivät yliluonnollisen suurilta virtuaalitodellisuudessa. Valitettavasti en silloin tiennyt kolmio määrien vaikuttavan suorituskykyyn merkittävästi.

Seuraava askel oli vähentää dynaamisten varjojen piirtoetäisyyttä. Oletuksena piirtoetäisyys oli 20000 yksikköä. Päätin pudottaa piirtoetäisyyden 1500 yksikköön. Tämän lisäksi pudotin myös varjojen laatua puolella. Taulukosta 6 nähdään näiden muutosten vaikutus suorituskykyyn.

Taulukko 6. Varjojen laatua ja piirtoetäisyyttä vähennetty.

Suorituskyky	29,07 ms

Piirtokutsuja	1698 kpl
Kolmioita	10 409 501 kpl
Tavoite suorituskyky	11,1 ms

Seuraavaksi muutin 3D-mallien tarkkuustaso asetuksia ja piirtoetäisyyksiä. Näiden muutosten tarkoitus oli vähentää piirrettävien kolmioiden määrää ja siten parantaa suorituskykyä. Joissakin 3D-malleissa, kuten kivissä, päädyin ottamaan kaksi yksityiskohtaisinta tarkkuustasoa kokonaan pois käytöstä.

Kuva 8. Kiven tarkkuustasojen ero. Kolmioita oikeassa on 3 921 ja vasemmassa 470 kpl.

Kuvassa 8 näkyy tarkimman ja kolmanneksi tarkimman tarkkuustason eron. Ero ei ole kovin suurin ottaen huomioon, että oikeassa kuvassa on yli 3000 kolmiota vähemmän. Eikä eroa huomaa kentässä, sillä missään kivessä ei ole käytössä yksityiskohtaisempia tarkkuustasoja. Taulukosta 7 näkyy näiden muutosten vaikutuksen suorituskykyyn.

Taulukko 7. Piirtoetäisyyksien ja tarkkuus tasojen muuttaminen

Suorituskyky	27,06 ms
Piirtokutsuja	1578 kpl
Kolmioita	7 974 088 kpl
Tavoite suorituskyky	11,1 ms

Tässä vaiheessa alkoi olemaan varmaa, että 90 FPS:n tavoitetta en saavuta mitenkään. Ideat ja aika alkoi olemaan lopussa ja lisäksi kiireellisempi asiakasprojekti tuli tämän projektin rinnalle, johon suurin osa ajastani meni. Tämän projektin loppuun hoitivat harjoittelijat, jotka eivät osanneet tehdä suorituskykymittauksia, joten tästä eteenpäin minulla ei ole tilastoja esimerkiksi kolmiomääristä ja suorituskyvystä. Viimeinen teko ennen siirtymistä toiseen projektiin oli vaihtaa kaikki puiden 3D-mallit yksinkertaisempiin malleihin.

Kuva 9. Puu mallien vertaus.

Kuvassa 9 vasemmalla on alkuperäinen puu malli ja oikealla uusi. Kolmioita alkuperäisessä oli 34 435 ja uudessa vain 6 606 kappaletta. Lisäksi materiaali paikkoja uudessa on neljä ja alkuperäisessä oli yhdeksän. Materiaali paikkojen määrä lisää piirtämiseen tarvittavia piirtokutsuja.

Projekti muuttui myös hieman, kun olin siirtynyt pois siitä. Alkuperäisessä suunnitelmassa pelaaja sai vapaasti liikkua alueella, mutta nyt pelaajaa kuljetettiin automaattisesti ennalta määrättyä polkua pitkin. Tämän tyylinen pakotettu liike on juuri mitä virtuaalitodellisuudessa tulisi välttää, sillä se aiheuttaa hyvin helposti pahoinvointia. Lisäksi vuorokauden, säätilan ja vuodenajan vaihtuminen tuli uusina ominaisuuksina mukaan. Jälkimmäinen aiheutti uusia ongelmia suorituskykyyn, sillä vesi- ja lumisade efektit tuli mukaan. Päivää ennen kuin projektin piti olla valmis, tein viimeisen optimoinnin projektiin. Otin käyttöön automaattisen resoluution säädön virtuaalitodellisuudessa. Tämä pudottaa resoluutiota, mikäli sovellus ei pääse 11,1 millisekuntiin. Tämä ei ole hyvä tapa parantaa suorituskykyä, sillä resoluution pudotuksen käyttäjä näkee selkeästi, mutta muuta vaihtoehtoa ei tässä vaiheessa enää ollut.

Taulukko 8. Lopullinen suorituskyky.

Suorituskyky	8,42 ms
Piirtokutsuja	1578 kpl
Kolmioita	5 848 227 kpl
Tavoite suorituskyky	11,1 ms

Taulukossa 8 on ilmoitettu lopullinen suorituskyky. Huomioitavaa lopullisessa suorituskyvyssä on, että se on testattu uudemmalla Unreal Engine 4 -pelimoottorin versiolla, joten tulokset eivät ole suoraan verrannollisia keskenään. Projekti tehtiin 4.17 versiolla ja lopullinen testi on tehty 4.20 versiolla. Aiemmalla versiolla suorituskyky ei päässyt tavoitteeseen.

2.4 Yhteenveto

Optimointi on oleellinen osa pelikehitystä, sillä se vaikuttaa suoraan käyttäjän pelikokemukseen. Virtuaalitodellisuussovelluksissa optimointi nousee vielä enemmän esille, koska paljon vaihteleva tai matala FPS voi aiheuttaa joillekin henkilöille mm. pahoinvointia, huimausta ja päänsärkyä.

Optimointia tehdään yleensä kolmessa kategoriassa: prosessori, näytönohjain ja muisti. Prosessorin suorituskykyyn vaikuttaa eniten pelilogiikka sekä objektien määrä pelimaailmassa. Näytönohjaimen suorituskykyyn vaikuttaa mm. sovelluksen resoluutio, objektien materiaalit, valaistus sekä post process -efektit. Muisti voidaan vielä jakaa keskus- ja videomuistiin. Muistin osalta eniten suorituskykyyn vaikuttaa tekstuurien tarkkuus.

3 Case japanilaisen metsän visualisointi

Projekti saatiin aikaisemmin tehdyn virtuaalitodellisuussovelluksen ansiosta. Aikaisemmassa projektissa teimme pienen metsäalueen virtuaalitodellisuutta varten, jossa käyttäjä kuljetettiin metsän sisällä. Kuljetuksen aikana vuoden aika vaihtui kesästä talveen. Projekti tehtiin messuja varten, joten käytön yksinkertaistamiseksi käyttäjä ei voinut vapaasti liikkua metsän sisällä. Kyseinen projekti oli esittelyssä Japanissa tekniikkamessuilla. Siellä tämän projektin asiakkaalla heräsi mielenkiinto lähteä tekemään yhteistyötä japanilaisen metsän visualisoinnista. Tässä osiossa käydään läpi mm. projektissa käytetyt työkalut sekä asiakkaan antamat taustatiedot ja miten niitä hyödynnettiin. Lisäksi kuvataan, kuinka varsinainen toteutusta tehtiin Unreal Engine 4 -pelimoottorilla.

3.1 Projektin kuvaus

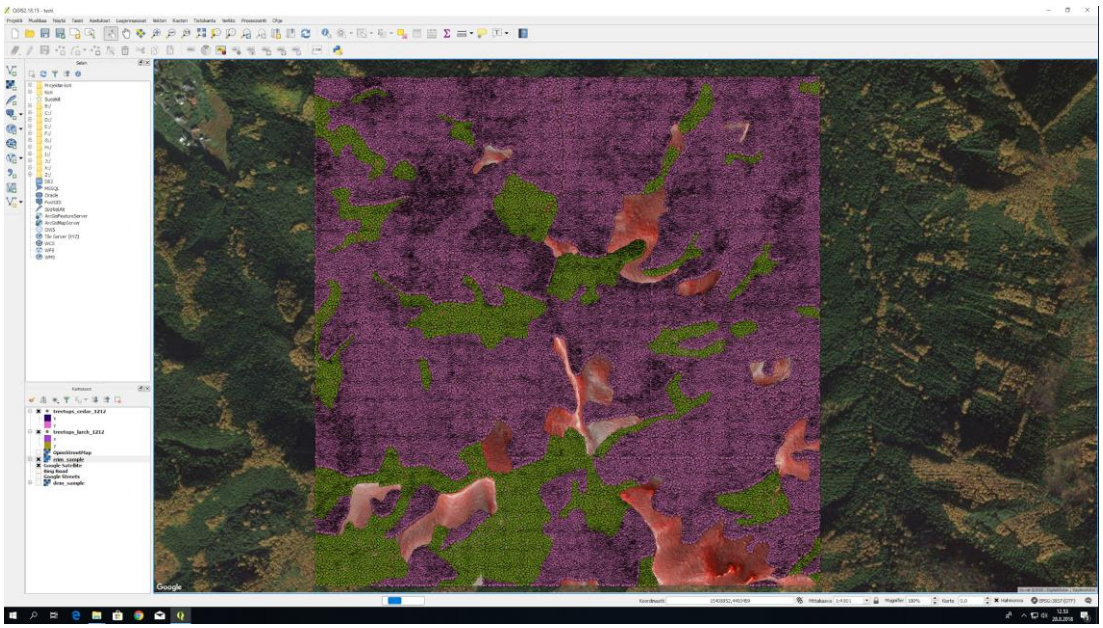
Projektin tavoitteena oli tehdä pieni kokeilu visualisointi 1 * 1Km alueesta metsää Nakanon kaupungin läheltä. Visualisoinnin tarkoituksena oli kuvata metsää alueella, jonne on hankala päästä oikeassa elämässä. Kyseinen alue onkin vuoristoa, jossa ei ole teitä. Asiakkaan tarkoituksena teettää pieni kokeilu, jonka jälkeen mahdollisesti visualisoida sitten isompia alueita. Asiakas ei halunnut virtuaalitodellisuusversiota, joten teimme tavallisen Windows-version lopullisesta sovelluksesta. Tämä helpotti optimointia myöhemmässä vaiheessa, koska ei sovelluksen suorituskyvyn ei tarvinnut ylittää 90 FPS:n rajaa.

Projektissa mukana oli kaksi henkilöä. Graafikko, joka teki puiden 3D-mallit, materiaalit ja tekstuurit. Lisäksi hän teki osan maatekstuureista. Toinen osa oli Unreal Engine 4 -pelimoottorin valmiita tekstuureja. Itselläni päätehtäviä tehtäviä oli ohjelmointi ja kentän luonti Unreal Engine 4 -pelimoottorissa. Ohjelmoinnin osalta tekemistä ei aluksi ollut paljoa, sillä muuta toiminnallisuutta ei tarvittu kuin hahmon liikkuminen. Projektissa käytettiin valmista Unreal Engine 4 -pelimoottorin pohjaa, jossa hahmo ja sen liikuttelu oli valmiiksi toteutettu. Tämä säästi aikaa muuhun tekemiseen. Kentän luonti piti sisällään mm. maaston luonnin, maaston materiaalin teon sekä puiden istutuksen.

Tärkeimpänä työkaluna oli Unreal Engine 4 -pelimoottorin 4.18.3 versio. Muita käytettyjä työkaluja oli Graafikon käyttämät SpeedTree-ohjelma puiden tekemiseen, PhotoShop-kuvankäsittelyohjelma kuvien muokkaamiseen, Substance Desinger tekstuurien tekoon. SpeedTree on 3D-mallinnusohjelma, joka on suunniteltu puiden ja kasvien mallintamiseen. Se teki puiden mallintamisesta nopeampaa, sillä se loi automaattisesti tarkkuustasot tehdyille puille.

Myöhemmin löytyi QGIS avoimen lähdekoodin paikkatieto-ohjelma, jolla voi käsitellä geologista tietoa, kuten esimerkiksi korkeusmalleja. Tämän ominaisuuden takia päädyimme käyttämään QGIS-ohjelmaa projektissa.

QGIS pystyi myös lukemaan asiakkaan antamat puu tiedot ja näyttämään jokaisen yksittäisenä pisteenä kartalla koordinaatin kanssa. Kuvassa 10 on QGIS näkymä visualisoitavasta alueesta.



Kuva 10. QGIS näkymä alueesta.

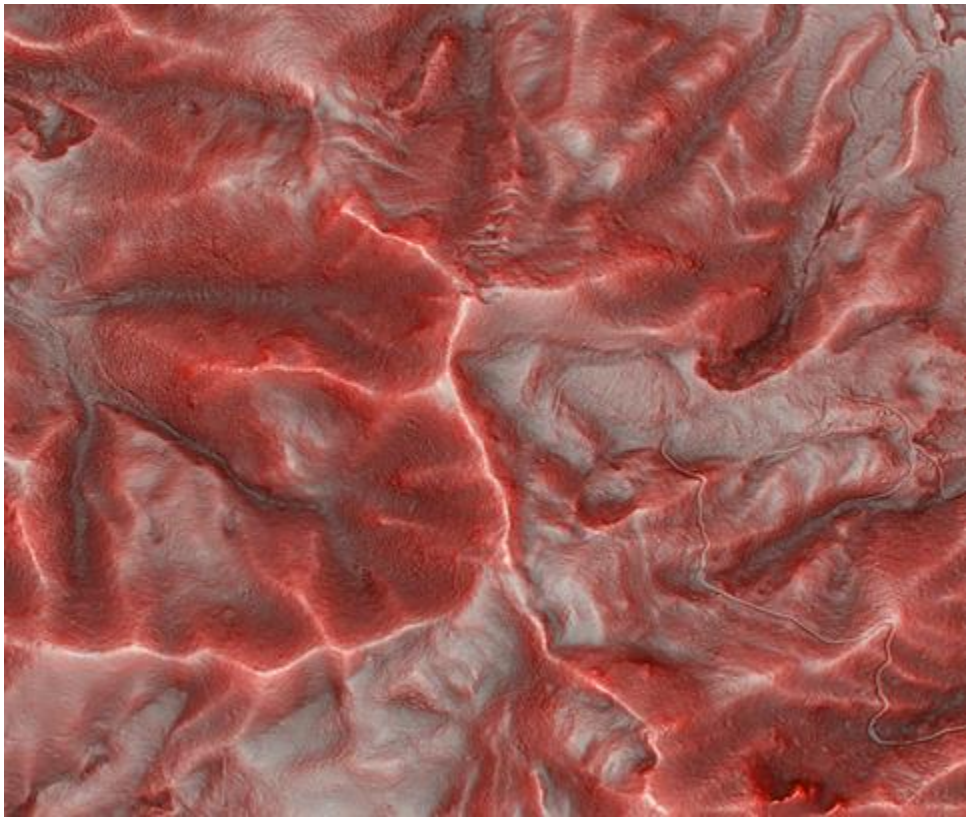
3.2 Toteutus

Tässä luvussa kuvataan projektin toteutus kokonaisuudessaan. Kuinka asiakkaan antamia taustatietoja hyödynnettiin. Lisäksi kuvataan kuinka maasto luotiin korkeusmallin pohjalta ja kuinka puut istutettiin. Korkeusmalli pitää sisällään maanpinna korkeuspisteet.

3.2.1 Taustatiedot

Asiakas toimitti meille tiedon puulajeista sekä myös tarkempia tietoja yksittäisistä puista. Korkeusmallista oli kaksi versiota. Alla oleva .tif eli tagged image file format kuva sekä .img päätteinen digital elevation model tai dem tiedosto. TIFF-kuvaa käytetään yleisesti kuvien varastoinnissa, sillä se mahdollistaa usean eri kuvan tallentamisen samaan TIFF-tiedostoon. (Technopedia 2018.) Jälkimmäistä emme saaneet auki millään käytössä

olevalla ohjelmalla, joten kokeilimme hyödyntää TIFF-kuvaa maaston luonnissa. Kuvassa 11 näkyy asiakkaan toimittama TIFF-kuva.



Kuva 11. Asiakkaan toimittama korkeusmalli visualisoitavasta alueesta.

Kattavat puusto tiedot asiakas toimitti kahdesta puulajista, setristä ja lehtikuusesta. Alueella kasvaa myös lehtipuita, mutta niitä ei ollut tarkoitus visualisoida tarkasti. Asiakkaalla ei myöskään ollut minkäänlaista tietoa lehtipuista.

	A	B	C	D	E	F	G
1		x,N,19,11	y,N,19,11	Height,N,19,11	CrownRatio,N,	DBH,N,19,11	Volume,N,19,11
2	0	-7721.75	79164.25	5.15	21.36	13.5660578511	0.0357497048
3	1	-7720.25	79166.25	5.2	28.08	13.5660578511	0.0361125674
4	2	-7720.25	79164.75	5.27	41.37	19.5503483946	0.0750658029
5	3	-7814.25	79214.25	5.52	44.38	26.3055366338	0.1292823966
6	4	-6806.25	80063.25	6.05	49.59	17.0885993875	0.0665728129
7	5	-7833.75	79181.75	6.63	54.45	15.4037607322	0.0597461012
8	6	-7950.25	78948.75	7.05	11.06	11.9475970069	0.0395809787
9	7	-7808.75	79175.25	7.41	59.24	35.2470053937	0.2943553228
10	8	-7838.25	79186.25	7.68	29.3	18.9577229939	0.1047448304
11	9	-7834.25	79176.75	7.84	61.1	20.4146673753	0.1237805084
12	10	-7446.75	79598.25	7.88	58.12	16.7619360681	0.0844879319
13	11	-7946.75	78951.75	8.16	57.35	16.0936916756	0.0808985465
14	12	-7821.25	79208.75	8.17	62.67	21.2521451117	0.1398502639
15	13	-7949.75	78950.25	8.22	28.83	14.3225211562	0.0648348748
16	14	-7715.75	79167.75	8.29	63.45	24.62676153	0.1798775436
17	15	-7943.75	78952.25	9.02	59.09	18.3507380557	0.1162460644
18	16	-7479.25	79153.75	9.26	54.86	25.115055623	0.210893015
19	17	-7133.25	79397.25	9.42	53.93	15.4037607322	0.0862474394
20	18	-7836.25	79182.75	10.09	57.68	22.8565177022	0.1950302211
21	19	-7911.25	78965.75	10.29	61.42	25.115055623	0.2371055446
22	20	-7613.75	79236.25	10.38	21	17.0885993875	0.1170422595
23	21	-7824.75	79216.25	10.42	69.77	32.3427498467	0.3729427657
24	22	-7990.25	79082.25	10.47	31.04	9.6524191413	0.0386189372

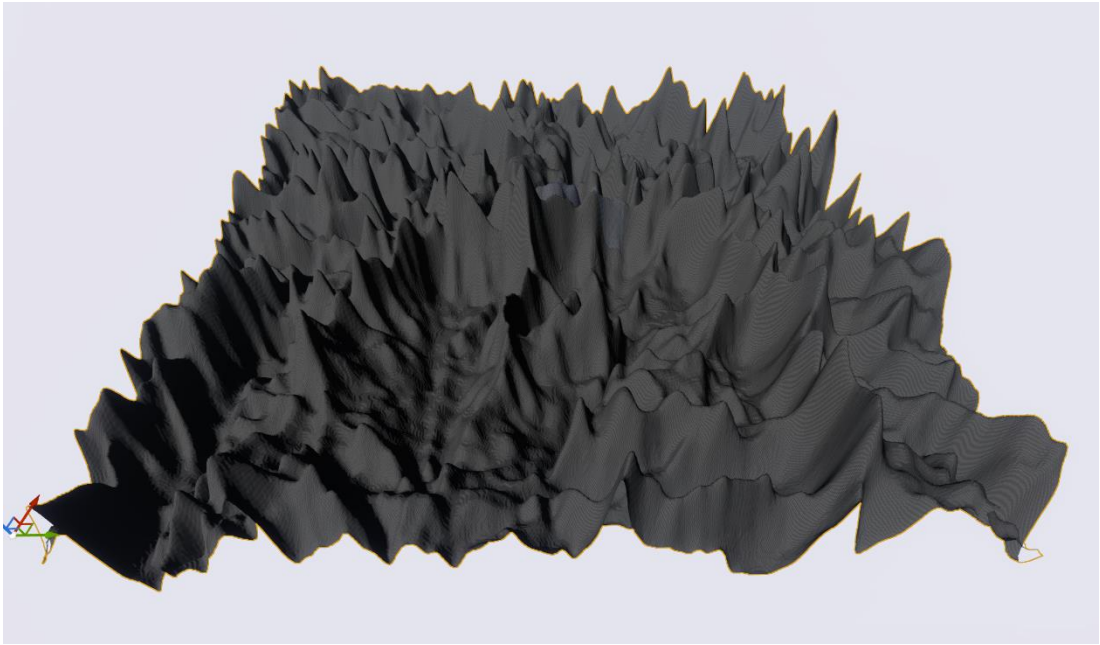
Kuva 12. Esimerkki puu tiedoista.

Kuvassa 12 on ensiksi puun indeksi, sitten tulee X- ja Y-koordinaatti ja puun korkeus metreinä. CrownRatio tarkoittaa kuinka monta prosenttia puun korkeudesta on lehtien peitossa (DeYoung 2018). DBH eli diameter at breast height tarkoittaa puun läpimittaa rinnan korkeudelta (Wikipedia 2018). Tässä tapauksessa läpimitta on ilmoitettu senteissä. Viimeinen sarake on yhden puun tilavuus. Näistä tiedoista käytimme koordinaatteja sekä korkeutta. CrownRatio tehtiin keskiarvona, sillä muuten jokaiselle puulle olisi joutunut tekemään oman 3D-mallin, joka olisi ollut käytännössä mahdotonta. Noin sadantuhannen puun mallintaminen olisi vienyt liian paljon aikaa. Samasta syystä myös puun paksuus oli keskiarvo.

Istutusvaiheessa pilkoimme puusta saadut tiedot korkeuden perusteella pienempiin paloihin. Esimerkiksi setreistä saadut tiedot pilkottiin kuuteen osaan: 5-11, 12-16, 17-21, 22-27, 28-31 ja yli 32 metriset puut. Tämä siksi, koska graafikko oli tehnyt 6 3D-mallia setri puista, viiden metrin välein alkaen kymmenestä metristä. Näin kaikki 5-11 metriset puut käyttivät kymmenen metrin 3D-mallia. Kuudella 3D-mallilla saatiin metsään enemmän variaatiota ja sitä kautta näyttämään luonnollisemmalta.

3.2.2 Maaston luonti

Asiakkaan toimittama korkeusmalli ei soveltunut käytettäväksi Unreal Engine 4 -pelimoottorin sisällä. Unreal Engine 4 -pelimoottori käyttää harmaasävy PNG-kuvia korkeuden luomisessa (Epic Games 2018e). PNG eli portable network graphics on yleisesti käytetty kuvaformaatti. PNG-kuvat tukee läpinäkyvyys kanavaa, joten niitä käytetään usein tekstuureina Unreal Engine 4 -pelimoottorissa Pyysimme asiakasta toimittamaan harmaasävy PNG-korkeusmallin, mutta valitettavasti asiakas ymmärsi väärin ja teki harmaasävy version samasta kuvasta minkä he olivat jo meille toimittaneet. Testasimme kuitenkin luoda maaston Unreal Engine 4 -pelimoottorissa tämän kuvan avulla.

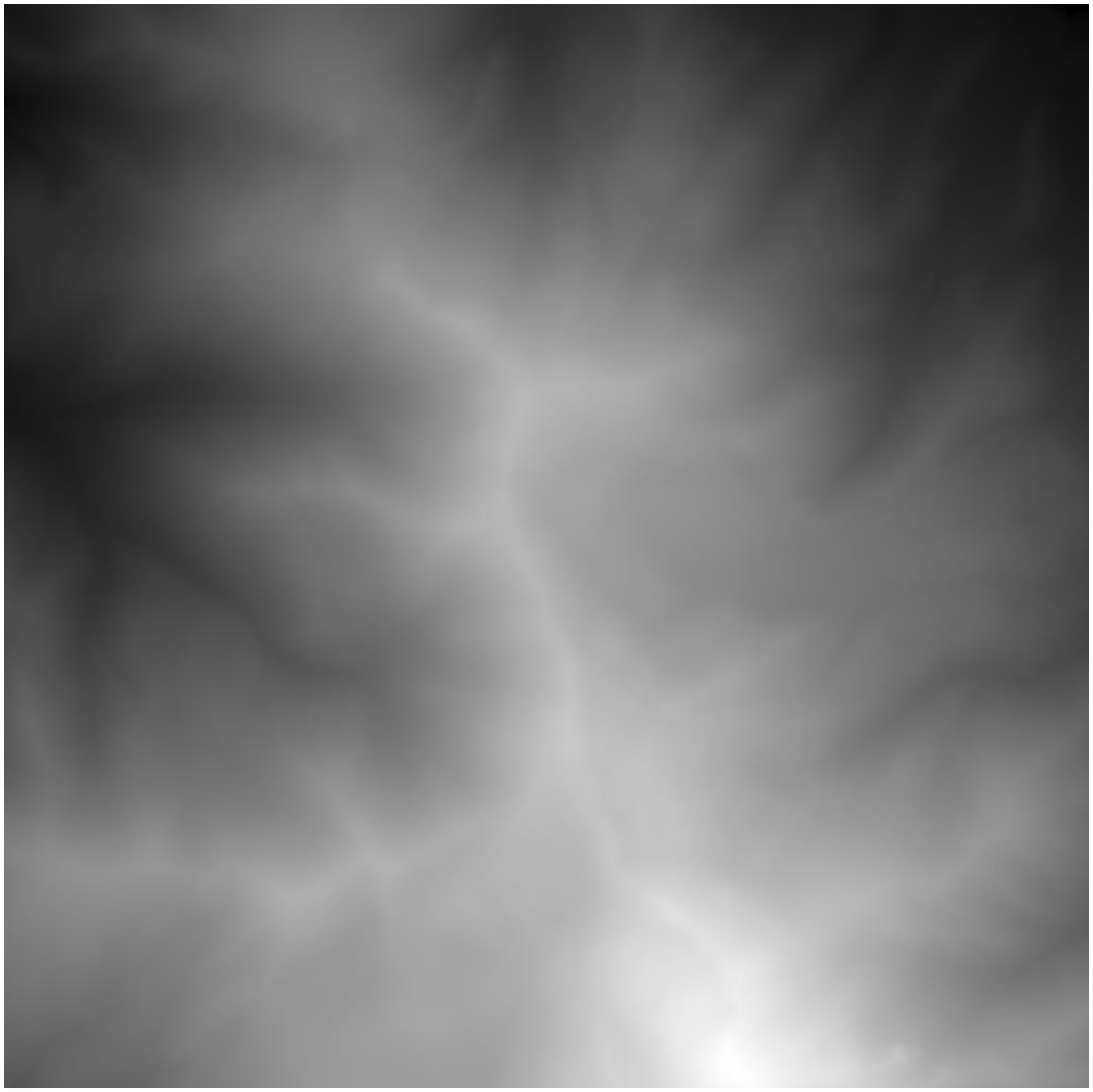


Kuva 13. Asiakkaan toimittamalla PNG-kuvalla luotu maasto.

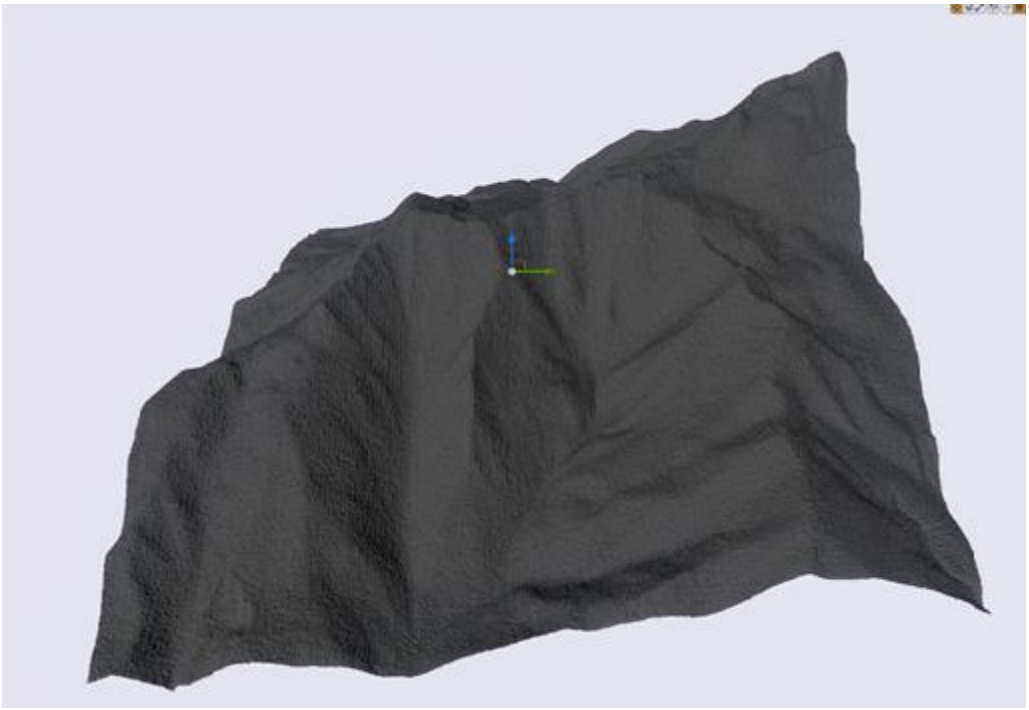
Meillä ei ollut tarkkaa sijaintia visualisoitavasta alueesta, joten emme voineet varmistaa näyttääkö kuvassa 13 oleva maasto samalta, mitä esimerkiksi Google Earth luoma 3D-maasto alueesta. Kuvan 13 maasto ei kuitenkaan näytä kovin luonnolliselta, joten oli turvallista olettaa, että asiakkaan toimittamalla korkeusmallilla luotu maasto ei näytä oikealta.

Oikea korkeusmalli saatiin käyttämällä QGIS-ohjelmaa. Asiakkaan antama DEM-tiedosto saatiin avattua lopulta QGIS-ohjelmalla, joka konvertoi sen

yleisesti käytettyyn korkeusmalli muotoon. QGIS-ohjelma antoi myös alueen koordinaatit, joiden avulla pystyimme tarkistamaan maaston muodon Google Earthin avulla. Tällä kertaa lopputulos oli oikea. Kuvassa 14 QGIS-ohjelmalla saatu korkeusmalli ja kuvassa 15 kyseisellä korkeusmallilla luotu maasto.



Kuva 14. QGIS ohjelman konvertoima korkeusmalli

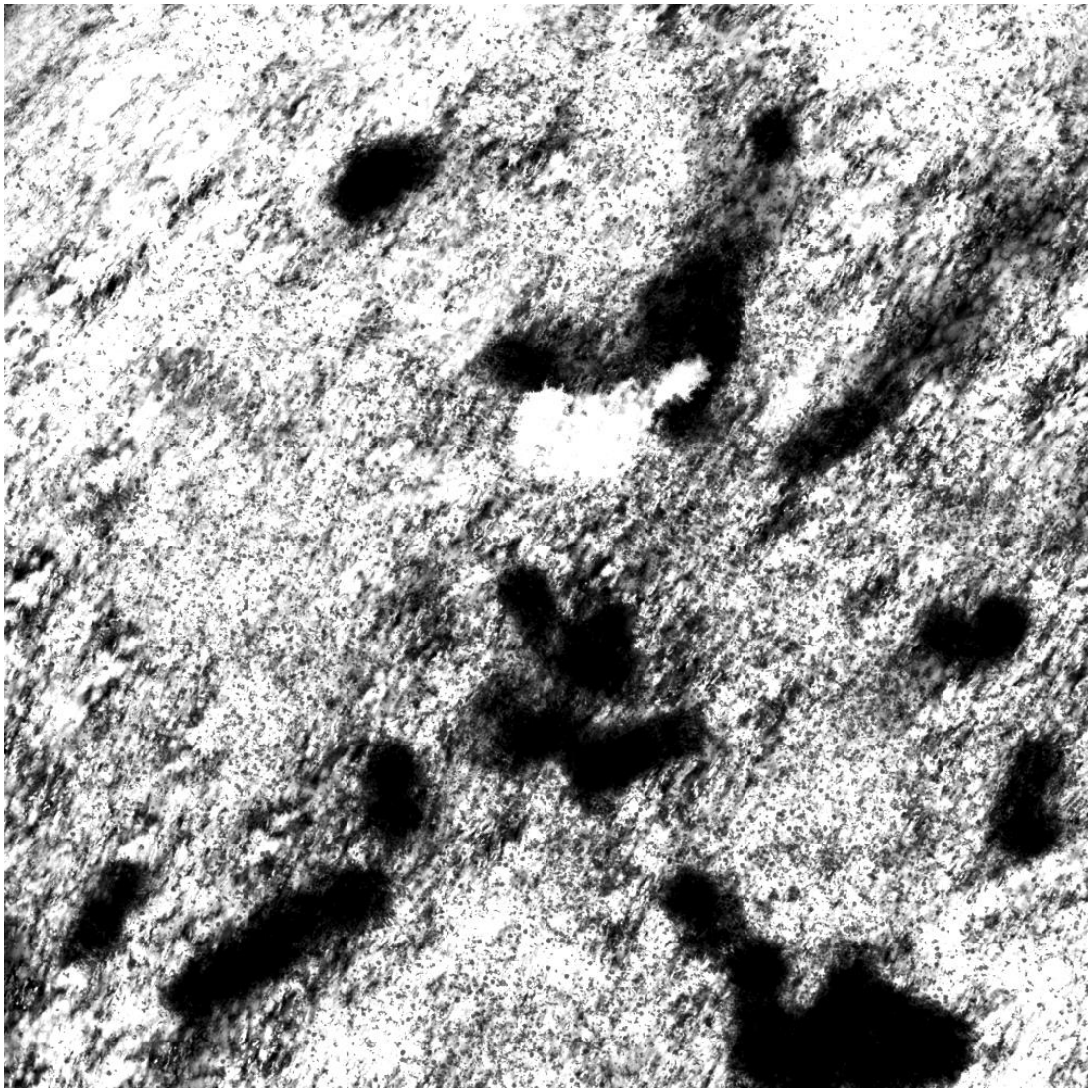


Kuva 15. Konvertoidulla korkeusmallilla luotu maasto.

Nyt kun tiesimme, miten saadaan oikea korkeusmalli alueesta, niin seuraava askel oli luoda maasto oikeassa mittakaavassa. Oletuksena maastoa luodessa Unreal Engine 4 -pelimoottorissa, maaston X- ja Y-skaala on 100 Unreal yksikköä. Tämä tarkoittaa sitä, että maaston tiheys on 1 verteksi metriä kohden (Epic Games 2017i). Korkeusmallia käytettäessä Unreal Engine 4 -pelimoottori luo yhden verteksin kuvan pikseliä kohden (Epic Games 2017j). Mikäli korkeusmallin resoluutio on 100*100 pikseliä, niin oletuksena maaston koko on silloin 100*100 metriä. Asiakas kertoi alueen olevan 1000*1000 metriä, joten korkeusmallin resoluution tulisi olla silloin 1000*1000 pikseliä. Lähin suositeltu koko on kuitenkin 1009*1009 pikseliä, joten päädyimme käyttämään sitä. (Epic Games 2017k.) Maaston Z-akselin korkeus on oletuksena 512 metriä, matalin kohta -256 metriä ja korkein kohta 256 metriä. Kaava oikean Z-skaalan laskemiselle on: (maaston korkeus/512) *100. Maastoa tehdessä kuitenkaan vielä tiennyt tätä, joten päädyin mittaamaan matalimman ja korkeimman kohdan eron Unreal Engine 4 -pelimoottorissa silmämääräisesti kuution avulla. Kuution Z-skaalaksi oli asetettu asiakkaan ilmoittama maaston korkeus.

3.2.3 Maaston materiaalin teko

Maaston materiaali oli hyvin yksinkertainen tässä projektissa, sillä asiakkaan suurin prioriteetti visualisoinnissa olivat puut. Meillä ei myöskään ollut kuvia tai tietoa japanilaisen metsän pohjasta. Päädyimme tästä syystä käyttämään kolmea tekstuuria, sammal, havumetsän pohja sekä lehtimetsän pohja. Materiaalin pohjakerroksena käytimme sammalta, jonka päälle tuli maskin avulla automaattisesti maalattu havumetsän pohjakerros. Käytetyn maskin graafikko teki PhotoShop-ohjelmalla. Lehtimetsää alueella oli hyvin vähän, joten se kerros maalattiin käsin. Kuvassa 16 on maski havukerrokselle. Valkoisille alueille tulee havutekstuuri. Isot mustat alueet ovat lehtimetsää.



Kuva 16. Havukerros maski.

Aluskasvillisuuden tekemiseen Unreal Engine 4 -pelimoottorista löytyy GrassMap-materiaalinode. Materiaalinodet ovat Unreal Engine 4 -pelimoottorin materiaalieditorin käyttämä visuaalinen ohjelmointikieli varjostimia varten. Tämä node tulee maaston materiaaliin, jossa voi määritellä eri kerrokselle erilaisen ruohotyypin. Ruohotyypin sisälle voi asettaa useita 3D-malleja ja antamaan niille tiheysparametrin. Näin aluskasvillisuus tulee automaattisesti halutulla tiheydellä halutulle maaston materiaalin kerrokselle. (Epic Games 2018f.) Tällä nodella pystyisi tekemään myös puita, mutta niiden sijaintia ei voi kontrolloida ruohotyypin kautta.

Myös aluskasvillisuudesta ei ollut mitään tietoja meillä, eikä asiakas halunnutkaan visualisoida sitä tarkasti. Päädyimme siksi käyttämään Unreal Engine 4 -pelimoottorin omia ruoho- ja pensas- 3Dmalleja ajan säästämiseksi. Kuvassa 17 näkyy GrassMap-materiaalinodella toteutettua aluskasvillisuutta.

Kuva 17. Aluskasvillisuutta metsän sisällä.

3.2.4 Puiden istutus

Unreal Engine 4 -pelimoottori tarjoaa kaksi työkalua puiden istutukseen. Ensimmäinen on kasvityökalu, jolla puut maalataan maastoon käsin.

Työkalussa voi asettaa useita kasvityyppejä sekä jokaisen kasvin istutusparametreja voi säätää, kuten esimerkiksi istutustiheyttä sekä kasviinstanssien etäisyyttä toisistaan. Työkalussa valitaan halutut kasvityypit sekä pensselin koko ja sen jälkeen istutus maastoon tapahtuu hiirellä maalaamalla. (Epic Games 2018g.) Halutessaan voi myös valita, että kasvit voi istuttaa toisten 3D-objektien päälle, kuten esimerkiksi rakennusten.

Toinen työkalu on procedural foliage spawner, joka tekee puiden istuttamisen automaattisesti sille annettujen parametrien mukaan. Parametreja tässä ovat esimerkiksi siemenpuiden lukumäärä, kuinka monta siementä yksi siemenpuu tekee, kuinka monta kertaa siemeniä syntyy, kuinka kauas siemenet leviävät, maksimi-ikä, ja puun ikä syntyessään. (UnrealEngine 2015.) Tällä työkalulla on nopeaa tehdä laaja metsäalue esimerkiksi videopeliin. Oikean metsän visualisointia varten tämä ei kuitenkaan sovellu, sillä kyseisiä parametreja säätämällä on mahdoton kontrolloida puiden sijaintia. Lukumäärän kontrollointi on myös hyvin vaikeaa.

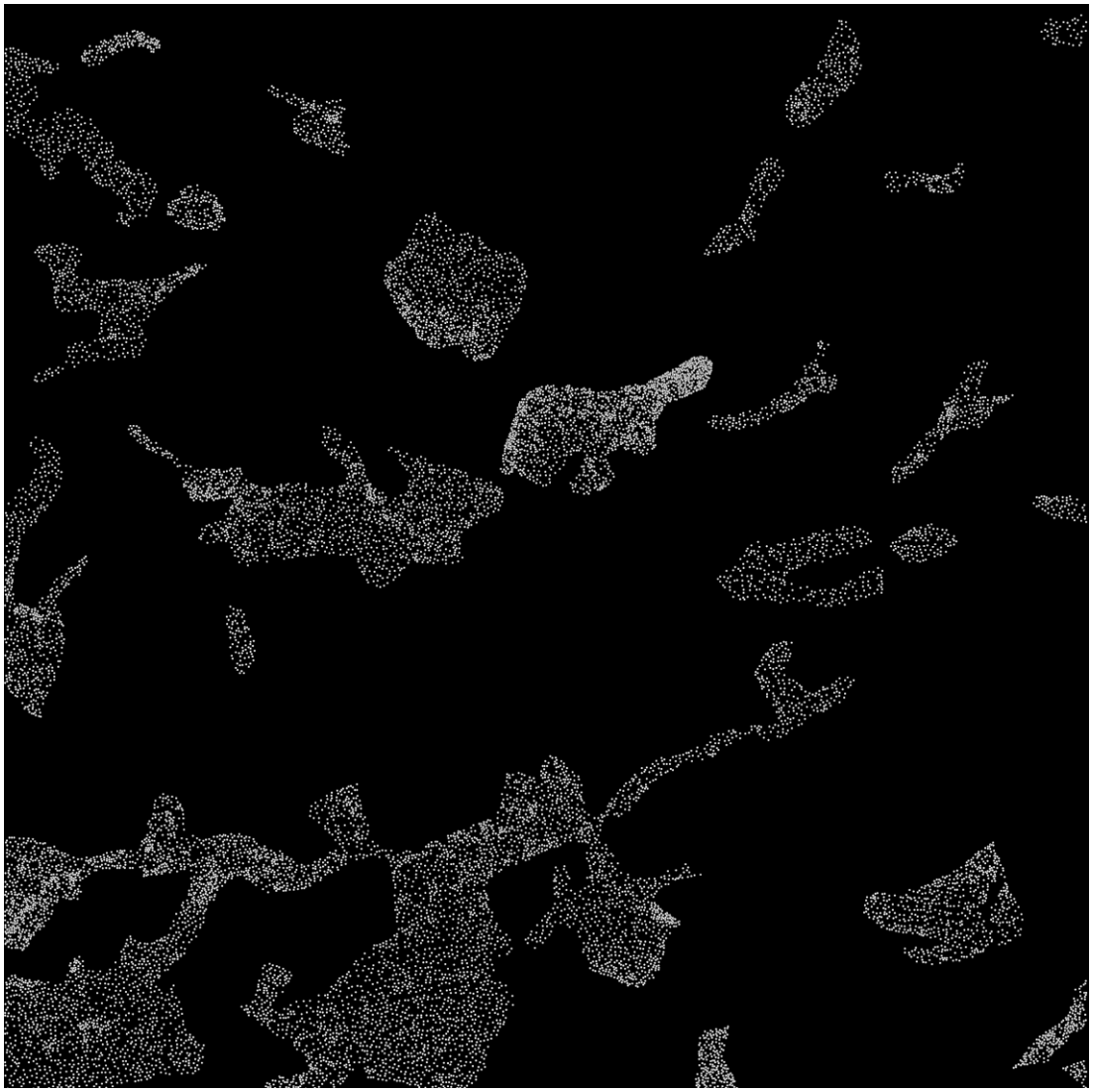
3.2.5 Puiden istutus, toteutus 1

Unreal Engine 4 -pelimoottorin oma kasvityökalu oli ensimmäinen vaihtoehto istuttaa puut maastoon. Pian kuitenkin huomasin, että puiden istutukseen ei voi kovin paljoa vaikuttaa. Suurin ongelma oli saada puiden lukumäärät edes suurin piirtein samoiksi kuin asiakkaan antamissa tiedoissa. Tämä onnistui puutyypin kasvutiheyttä ja etäisyyttä säätämällä yrityksen ja erehdyksen kautta. Kyseinen tapa oli hyvin työläs, sillä puutyyppejä oli yhteensä 13. Seuraava ongelma oli saada puut suurin piirtein oikeille paikoille. Tapoja tähän olisi esimerkiksi silmämääräinen istutus referenssikuvien avulla. Kuvia ei ollut, joten tämä tapa ei onnistunut. Toinen tapa olisi tehdä maaston materiaaliin satelliittikuvatekstuuri alueesta, jota käyttää istutuksessa. Ongelma oli saada satelliittikuva tarkalleen samasta paikasta kuin korkeuskartta. Viimeinen tapa oli istutus mustavalkomaskien avulla. Ideana oli tehdä maastomateriaaliin oma kerros lehtikuusille ja setreille ja käyttää QGIS-ohjelmasta otettua kuvaa puiden sijainneista maskina. Kasvi-työkalussa pystyy puutyypille asettamaan

maastomateriaalikerroksen, johon puut istutetaan. Tämä oli tapa millä ensimmäinen toteutus tehtiin. Kuvassa 18 istutetaan puita maalaamalla. Ruskeat pisteet merkkaavat yhden puun paikkaa.

Kuva 18. Puiden istutus maalaamalla.

Maski saatiin käyttämällä QGIS-ohjelmaa, johon pystyi tuomaan asiakkaan antamat puiden sijaintitiedot. Ohjelma sitten näytti jokaisen puun omana pisteenä koordinaattien perusteella. Näitä pisteitä käytettiin maskin tekemiseen siten, että valkoinen piste merkkasi puun sijaintia ja musta puutonta aluetta.



Kuva 19. Maski lehtikuusien istuttamiselle.

Maskin teko riittävällä resoluutiolla oli erittäin työlästä. Projektissa ollut graafikko otti kuvakaappauksia puiden sijanneista QGIS-ohjelmalla ja yhdisti ne PhotoShop-ohjelmalla yhdeksi kuvaksi. Kuvasta tehtiin mustavalkoinen ja resoluutio pudotettiin 1009:ään, jotta se vastaisi maaston kokoa. Kuvassa 19 on tämän prosessin lopputulos. Tästä tullut maski teki puiden istuttamisesta nopeampaa ja sillä sai suurin piirtein oikean määrän puita oikealle paikalla. Ongelma tuli siinä, että puiden sijaintia kuvaavat maastokerrospisteet jäivät liian suuriksi, joten puut monesti olivat 2-4 puun kasoissa, mikä visuaalisesti ei näyttänyt hyvältä. Kuvassa 20 on kuvattu tämä ongelma. Jouduin käsin siirtämään puita irti toisistaan, kun teimme asiakkaalle videon projektista.

Käsin siirtäminen ei ollut järkevä ja tehokas vaihtoehto, joten päädyin tekemään oman istutusratkaisun.

Kuva 20. Istutusongelma.

3.2.6 Puiden istutus, toteutus 2

Edellä mainitulla tavalla metsästä ei tullut realistisen näköinen. Asiakas oli toimittanut meille tarkat lukumäärät puista. Lisäksi joka puulla oli X- ja Y-koordinaatti sekä korkeus ilmoitettuna. Päädyin tekemään oman puuistuttajan, joka hyödyntää kyseisiä tietoja. Aluksi piti miettiä, mitä ongelmia oman toteutuksen tekeminen toisi. Suurin ongelma oli suorituskyky. Mikäli jokainen puu olisi oma 3D-objekti, niin jokainen puu vaatisi oman piirtokutsunsa. Puita

kentässä oli noin 100 000 kappaletta yhteensä. Ratkaisuksi tähän oli sama komponentti, jota Unreal Engine 4 -pelimoottorin omat työkalut käyttävät, eli hierarchical instanced static mesh. Tämä komponentti luo halutun määrän instansseja sille asetetusta puu 3D-mallista. Ero tavallisessa 3D-objektissa ja instanssissa on, että kaikki instanssit samasta 3D-mallista voidaan piirtää yhdellä piirtokutsulla. Instansseilla voi kuitenkin olla uniikki sijainti, skaala sekä tarkkuustaso. (Epic Games 2017l).

Toinen ratkaistava ongelma oli, miten muuttaa asiakkaan tiedoissa oleva koordinaatisto Unreal Engine 4 -pelimoottorin koordinaatistoon sopivaksi, jotta puut menevät oikeille paikoilleen. Asiakas ilmoitti visualisoitavan alueen kooksi 1 * 1 km. Pelimoottoriin vietävän korkeus kartan kooksi tuli siis 1009 x 1009 pikseliä (Epic Games 2017k). Puiden koordinaatiston väli molemmissa X- ja Y-akselilla oli 1220 yksikköä. Jotta puut menisivät oikeille paikoilleen, niin tämä kokoero tuli ottaa huomioon istutuksessa. Koordinaatisto piti myös muuttaa lähtemään nollasta, jotta sen saisi helpommin Unreal Engine 4 -pelimoottorin koordinaatistoon. Tämä saatiin siten, että alueen kulmapisteen koordinaatista vähennetään puun oma koordinaatti. Valitsin alueen vasemman yläkulman lähtöpisteeksi. X-koordinaatti sille oli -8000 ja Y-koordinaatti 80100. Esimerkkinä puu jonka koordinaatit ovat -7998,75 ja 80097,25. Vähennyslaskun jälkeen saadaan koordinaateiksi -1,25 ja 2,75. Koordinaatteihin lisättiin maaston vasemman kulmapisteen sijainti, jotta puiden istutus alkaisi sieltä.

Aluksi tein yhden puuistuttajan hoitamaan kaikkien puutyyppien istutuksen. Ideana oli, että puudata taulukossa olisi yksi solu puutyypille. Istutuksen tekevässä For silmukassa asetettaisiin instanssin 3D-malli sen mukaan. Ongelma tuli vastaan, kun malli vaihtui, niin se vaihtui jokaiselle puulle, myös jo kentässä oleville. Nyt oli selvää, että jokainen eri puu tyyppi vaatisi oman istuttajan. Istuttajia tuli 13 kokonaisuudessaan. Tässä ilmeni toinen ongelma, koska puuistuttaja oli valittuna Unreal Engine 4 -pelimoottorissa, niin sitä mukaan kuin puita ilmestyi, niin pelimoottori korosti ne ja monta korostettua instanssia kulutti koneen RAM-muistin ja pelimoottori kaatui. Ratkaisu oli tehdä manageriluokka puuistuttajille, joka käskee istuttajia aloittamaan puiden

istutuksen. Näin ei tarvitse yksittäisen puuistuttajan olla valittuna pelimoottorissa, vaan istutus kaikissa eri istuttajissa tapahtui yhdellä napin painalluksella eikä kaatumisongelmaa enää ollut.

Puutiedot oli lisätty taulukoihin, joista istuttaja kävi ne läpi. Taulukoista löytyi X- ja Y-koordinaatti sekä korkeus. Lisäksi tarvittiin Z-koordinaatti, eli mille korkeudelle puu tulisi. Tätä arvoa ei ollut missään asiakkaan antamissa tiedoissa. Päädyin tekemään istutuksen yhteydessä Line tracen korkealta suoraan alaspäin. Line trace käytännössä tekee kuvitteellisen viivan kahden pisteen välille ja se tunnistaa, mikäli viiva osuu johonkin toiseen objektiin. Törmäyspisteen koordinaatin saa otettua talteen ja tätä päädyin hyödyntämään puun Z-koordinaatissa. Tässä tapauksen viivan lähtöpiste oli puun X- ja Y-koordinaatti taulukosta ja Z-koordinaatti oli 40 000 yksikköä. Loppupiste oli muuten sama, mutta Z-koordinaatti oli -40 000 yksikköä. Näillä arvoilla varmistin, että viiva alkaa maaston yläpuolelta ja päättyy maaston alapuolelle.

Puun korkeus tietoa käytettiin instanssin skaalan Z-arvon asettamiseksi. Skaala tarkoittaa 3D-mallin kokoa Unreal Engine 4 -pelimoottorissa. Skaalaa voidaan muuttaa X-, Y- ja Z-akselilla. Oletusarvona jokaisessa on 1,0 yksikköä. Skaala laskettiin puun todellisella korkeudella taulukosta jaettuna käytettävän 3D-mallin korkeudella. Esimerkiksi jos puun korkeus oli 8 metriä ja 3D-mallin korkeus 10 metriä, niin skaalaksi tulisi tällöin 0,8 yksikköä. Läpimittaa rinnan korkeudella olisi varmaan voinut hyödyntää vastaavasti puun X- ja Y-skaalan suhteen, mutta ajatus tästä tuli turhan myöhään projektin aikana, joten päädyin pitämään ne arvot oletuksena.

3.3 Projektin päätös

Projekti oli kaikin puolin onnistunut ja asiakas oli tyytyväinen lopputulokseen. Projektin lopputulos päätyi jopa esitettäväksi Suomen suurlähetystössä Japanissa. Japanin suunnalta oli myös halua jatkaa yhteistyötä vastaavien projektien merkeissä.

Itse voin olla myös tyytyväinen lopputulokseen. Toteutimme kaikki asiakkaan toiveet projektissa. Optimointia tässä projektissa ei ollut tarve tehdä, sillä 3D-mallit tehtiin tätä projektia varten. Kokemuksen myötä tiesimme pitää 3D-mallien kolmiomäärät matalina. Asiakkaalla ei myöskään ollut tavoitetta suorituskyvylle. Parannettavaa toki myös jäi. Esimerkiksi tuo itse tehty puun istutus työkalu olisi voinut olla paremmin toteutettu. Suurin ongelma sen kanssa oli, jos vahingossa valitsit jonkin puun kentästä, niin Unreal Engine saattoi kaatua. RAM-muisti loppui koska tuli liian monta instanssia aktiiviseksi samaan aikaan.

4 Yhteenveto

Mitkä asiat vaikuttavat suorituskykyyn optimoitavassa alueessa? Suorituskykyyn vaikuttavat käytettyjen 3D-mallien kolmiomäärä, sillä mitä enemmän kolmioita on, niin sitä enemmän työtä näytönohjain joutuu tekemään. Puiden lukumäärä ja sitä kautta piirtokutsujen määrä vaikuttaa merkittävästi suorituskykyyn.

Mitä erityistä pitää ottaa huomioon virtuaalitodellisuus sovelluksessa suorituskyvyn suhteen? Suorituskyky on tärkeää virtuaalitodellisuudessa, sillä se vaikuttaa käyttäjän kokemukseen huomattavasti. Heikko suorituskyky aiheuttaa herkemmin mm. pahoinvointia. Tästä syystä suorituskyky kannattaa huomioida heti projektin alkaessa. Mikäli mahdollista, niin assetit eli esimerkiksi 3D-mallit kannattaa tehdä itse. Näin niiden kolmiomäärää on helpompi kontrolloida. Muutenkin 3D-mallit ja materiaalit kannattaa pitää mahdollisimman yksinkertaisina heti alussa. Myöhemmässä vaiheessa niiden muuttaminen vie turhan paljon aikaa. Joissakin tilanteissa joutuu käyttämään ostettuja assetteja, mutta niiden valinnassa kannattaa huomioida suorituskyky myös. Helposti kaupasta tulee valittua hienoimman näköinen puu, mutta jos siitä johtuva heikko suorituskyky aiheuttaa pahoinvointia, niin käyttäjä ei kykenen nauttimaan siitä. Itse päädyin vaihtamaan projektissa alun perin olleet

puu 3D-mallit ja materiaalit yksinkertaisempaan projektin lopussa. Visuaalinen ilme hieman kärsi, mutta lopputulos oli kuitenkin kaiken kaikkiaan miellyttävämpi.

Miten prosessorin, näytönohjaimen ja muistin suorituskykyä mitataan Unreal Engine 4 -pelimoottorilla? Unreal Engine 4 -pelimoottorissa on kattavat profilointi työkalut. Stat unitgraph-konsolikomento näyttää eriteltynä näytönohjaimen viemän ajan yhden kuvan piirtämiseen sekä prosessorin viemän ajan yhden kuvan piirtämiseen. Tällä saadaan karkea kuva kumpi tarvitsee optimointia. Tarkempi näytönohjaimen suorituskyky mittaus saadaan ajamalla konsolikomento profileGPU. Tämä tekee lokitiedosto, jossa näkyy piirtokutsujen määrä, piirrettyjen kolmioiden määrä sekä esimerkiksi kuinka paljon aikaa menee post prosess-efekteissä tai varjojen piirtämisessä. Tarkka prosessorin suorituskykymittaus voidaan tehdä starfile ja stopfile konsolikomennoilla. Näillä saadaan lokitiedosto, josta näkee esimerkiksi kuinka paljon aikaa pelilogiikka vie. Muistin mittaamiseen on memreport-konsolikomento, josta näkee sovelluksen viemän keskusmuistin ja videomuistin määrän.

Kuinka hyödynnetään oikean maailman korkeusmalleja Unreal Engine 4 -pelimoottorissa? Unreal Engine 4 -pelimoottori tukee harmaasävy PNG-korkeusmalleja maaston luomisessa. Haasteena tässä projektissa oli saada vastaava korkeusmalli. Asiakas oli toimittanut digital elevation model tiedoston alueesta, jota emme aluksi saaneet auki millään käytössämme olevalla ohjelmalla. Ratkaisuksi löytyi QGIS niminen ohjelma, joka kykeni avaamaan DEM-tiedoston ja konvertoimaan sen PNG-kuvaksi.

Kuinka oikean maailman puusto tietoja hyödynnetään puiden istuttamissa Unreal Engine 4 -pelimoottorissa? Unreal Engine 4 -pelimoottorin valmiit kasvien istutustyökalut eivät toimineet tähän tarkoitukseen, joten jouduin toteuttamaan oman puuistuttajan. Puuistuttaja luki puiden koordinaatit taulukosta, konvertoi koordinaatin Unreal Engine 4 -pelimoottorin koordinaatistoon ja loi puuinstanssin. Puuinstansseissa käytettiin samaa

instanssi komponenttia, mitä Unreal Engine 4 -pelimoottorin omat kasvien istutustyökalut käyttivät.

Vaikka japanilaisen metsän visualisointia ei tehty virtuaalitodellisuutta varten, niin siinä kuitenkin hyödynnettiin aiemmin opittuja asioita suorituskyvyn suhteen. Projektissa oli mukana graafikko, joka teki puiden 3D-mallit ja materiaalit. Tästä syystä projektissa ei ollut suorituskyky ongelmia missään vaiheessa. Kattavaa profilointia eikä optimointia tarvinnut tehdä, joten säästimme aikaa.

Lähteet

Atwood, J. 2011. Fast Approximate Anti-Aliasing (FXAA). https://blog.codinghorror.com/fast-approximate-anti-aliasing-fxaa/. 2.11.2017.

DeYoung, J. 2018. https://openoregon.pressbooks.pub/forestmeasurements/chapter/5-4-lve-crown-ratio/. 20.9.2018.

Epic Games. 2017a. Creating and Using LODs. https://docs.unrealengine.com/latest/INT/Engine/Content/Types/StaticMeshes/HowTo/LODs/index.html. 2.11.2017.

Epic Games. 2017b. Forward Shading Renderer for VR. https://docs.unrealengine.com/en-us/Engine/Performance/ForwardRenderer. 3.11.2017.

Epic Games. 2017c. VR Profiling Interpretations and Considerations. https://docs.unrealengine.com/latest/INT/Platforms/VR/Profiling/Considerations/index.html. 2.11.2017.

Epic Games. 2017d. CPU Profiling. https://docs.unrealengine.com/en-us/Engine/Performance/CPU. 2.11.2017.

Epic Games. 2017e. GPU Profiling. https://docs.unrealengine.com/en-us/Engine/Performance/GPU. 7.11.2017.

Epic Games. 2017f. View Modes. https://docs.unrealengine.com/en-us/Engine/UI/LevelEditor/Viewports/ViewModes#shadercomplexity. 7.11.2017.

Epic Games. 2017g. Shadow Casting. https://docs.unrealengine.com/en-us/Engine/Rendering/LightingAndShadows/Shadows. 10.11.2017.

Epic Games. 2017h. Virtual Reality Best Practices. https://docs.unrealengine.com/en-us/Platforms/VR/ContentSetup. 10.11.2017.

Epic Games. 2017i. Landscape - Sizes and Height Guide. https://wiki.unrealengine.com/Landscape_-_Sizes_and_Height_Guide. 7.11.2017.

Epic Games. 2017j. Creating Landscapes. https://docs.unrealengine.com/en-us/Engine/Landscape/Creation. 7.11.2017.

Epic Games. 2017k. Landscape Technical Guide. https://docs.unrealengine.com/en-us/Engine/Landscape/TechnicalGuide. 7.11.2017.

Epic Games. 2017l. Foliage Instanced Meshes. https://docs.unrealengine.com/en-us/Engine/Foliage. 9.11.2017.

Epic Games. 2018a. FBX Static Mesh Pipeline. https://docs.unrealengine.com/en-us/Engine/Content/FBX/StaticMeshes. 14.11.2018.

Epic Games. 2018b. Performance and Profiling Overview.. https://docs.unrealengine.com/en-us/Engine/Performance/Overview. 14.11.2018.

Epic Games. 2018c. Slate Overview. https://docs.unrealengine.com/en-US/Programming/Slate/Overview. 14.11.2018.
Epic Games. 2018d. View Modes. https://docs.unrealengine.com/en-us/Engine/UI/LevelEditor/Viewports/ViewModes. 14.11.2018.
Epic Games. 2018e. Creating and Using Custom Heightmaps and Layers. https://docs.unrealengine.com/en-us/Engine/Landscape/Custom. 14.11.2018.
Epic Games. 2018f. Landscape Materials and the Grass Tool. https://docs.unrealengine.com/en-us/Engine/OpenWorldTools/Grass/QuickStart/3. 14.11.2018.
Epic Games. 2018g. Foliage Instanced Meshes. https://docs.unrealengine.com/en-us/Engine/Foliage. 14.11.2018.
Jukić, T. 2015. Draw calls in a nutshell. https://medium.com/@toncijukic/draw-calls-in-a-nutshell-597330a85381. 2.11.2017.
Klappenbach, M. 2018. Understanding and Optimizing Video Game Frame Rates. https://www.lifewire.com/optimizing-video-game-frame-rates-811784. 14.11.2018.
McKee, C. 2017. What Is Antialiasing. https://www.lifewire.com/antialiasing-pc-games-831764. 2.11.2017.
Nvidia. 2017. Technology. https://www.geforce.com/hardware/technology/txaa/technology. 2.11.2017
Owens, B. 2013. Forward rendering vs deferred rendering. https://gamedevelopment.tutsplus.com/articles/forward-rendering-vs-deferred-rendering--gamedev-12342. 5.11.2017.
Pettineo, M. 2016. A quick overview of MSAA. https://mynameismjp.wordpress.com/2012/10/24/msaa-overview/. 2.11.2017.
Schreibt, S. 2015. Render Hell - Book III. https://simonschreibt.de/gat/renderhell-book3/. 2.11.2017.
Technopedia. 2018. Tagged Image File Format. https://www.techopedia.com/definition/2093/tagged-image-file-format-tiff. 20.9.2018.
Tech Art Aid. 2017. UE4 Grpahics Profiling: All Categories Guide (Rendering Passes) https://www.youtube.com/watch?v=C3lumWdwHmA. 7.11.2017.
Tellez, B. 2014. How to improve game thread CPU performance in Unreal Engine. https://www.unrealengine.com/en-US/blog/how-to-improve-game-thread-cpu-performance. 14.11.2018.
Thoman, P. 2016. What ‘optimization’ really means in games. https://www.pcgamer.com/what-optimization-really-means-in-games/. 5.11.2017.
UnrealEngine. 2015. Foliage. https://www.youtube.com/watch?v=3ispyNxpRyg. 9.11.2017.
Wikipedia. 2017a. Shader.

https://en.wikipedia.org/wiki/Shader#Geometry_shaders. 5.11.2017.
Wikipedia. 2017b. 3D rendering.
https://en.wikipedia.org/wiki/3D_rendering. 5.11.2017.
Wikipedia. 2018. Diameter at breast height.
https://en.wikipedia.org/wiki/Diameter_at_breast_height. 20.9.2018.
Zeigler, B. 2014.Debugging and Optimizing Memory..
https://www.unrealengine.com/en-US/blog/debugging-and-optimizing-memory. 9.11.2017.